Telecommunications Management Network: A Novel Approach Towards its Architecture and Realisation Through Object-Oriented Software Platforms

George Pavlou

University College London

Thesis submitted for the degree of Doctor of Philosophy

March 1998

Abstract

The increasing sophistication of telecommunications infrastructures and, in particular, the introduction of broadband transmission and switching technologies together with advanced services that exhibit guaranteed Quality of Service characteristics, have necessitated the use of sophisticated management facilities. The Telecommunications Management Network (TMN) is being developed by the ITU-T as the framework to support the *open* management of telecommunications networks and services, exploiting the capabilities of emerging broadband technologies and harnessing their power. It projects a hierarchical distributed paradigm in which interactions are object-oriented (O-O) in information specifications terms. On the other hand, it is mainly a *communications* concept, and, as such, it does not address software realisation aspects. The TMN uses currently OSI Systems Management (OSI-SM) as its base technology.

The complexity of the combined OSI-SM/TMN architectural framework and the fact that non object-oriented approaches have been initially adopted for its realisation has resulted in doubts about its feasibility, implementability, performance and eventual deployment. This thesis proposes first a number of modifications and extensions to the TMN model and architecture. The modifications aim at the simplification of the overall framework and rely on the fact that this thesis shows that full scale OSI-SM/TMN technology is both feasible and performant. The extensions introduce distribution and discovery facilities through the OSI Directory and aim to support the TMN as a large scale distributed system.

The thesis subsequently shows that the inherent object-oriented aspects of the OSI-SM/TMN framework can be exploited through an object-oriented realisation model that hides protocol aspects, bears similarities to recently emerging O-O distributed systems frameworks and has good performance characteristics without requiring excessive computing resources. The environment in which these concepts were validated is known as OSIMIS (OSI Management Information Service) and can be used as a TMN *distributed O-O platform* that enables the rapid development and deployment of TMN systems. It predated similar products by some years and influenced a number of subsequent commercial developments.

Open Distributed Processing (ODP) has recently emerged as the theoretical framework for object-oriented distributed systems. The Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) can be seen as its pragmatic counterpart. Since the appearance of CORBA, a lot of research has addressed its use in TMN environments, because of its generality, better distribution paradigm and potentially better performance. Despite the relevant efforts though, there is no complete solution as yet that retains the full OSI-SM expressive power. This thesis proposes a solution that will make possible the seamless replacement of the OSI SM and Directory with CORBA as the base technology for the TMN.

To my family and friends

for their encouragement and support.

Acknowledgements

The work presented in this thesis was carried out over a long period of time while I was working initially as Research Associate, then as Research Fellow and finally as Senior Research Fellow and Lecturer at the Department of Computer Science, University College London. During that period, I was given the opportunity, support and freedom to do all this exciting research.

I would like to thank in particular the following people: Graham Knight, my first supervisor, who introduced me to the field of network management ten years ago, kept me on the rails during the formulation of this thesis and contributed the initial idea of generic software management environments; Prof. Peter Kirstein, my second supervisor, for giving me the freedom to undertake this type of research in projects for which he was the principal investigator and also for keeping me on the rails during this thesis; David Griffin, for the endless discussions we had on related subjects over the years and for commenting on the thesis; Dave Lewis, for instilling gradually to me his enthusiasm for open distributed processing technologies and for commenting on the thesis; a number of colleagues at UCL and elsewhere who contributed to the software environment that validated the concepts presented in this thesis - they are acknowledged separately; and finally the people at the University of Surrey, and in particular Prof. Barry Evans, who trusted me by offering a senior academic position - this has been the strongest stimulus for completing this thesis!

Last but not least I would like to thank my family, and in particular Christine for her forbearance over the years while I was spending endless late evenings and weekends in front of a pile of papers and the computer.

Acknowledgements to OSIMIS Contributors

Most of the concepts presented in this thesis have been validated in the OSI Management Information Service (OSIMIS) TMN platform. A number of colleagues at UCL and elsewhere have contributed to OSIMIS. The most important contributors are acknowledged below.

Graham Knight of UCL, my first supervisor, contributed initially the idea of a Generic Managed System, influenced the concept of the event-driven coordination mechanism, implemented the first version of the example UNIX object class, directed the security research work and in general influenced the OSIMIS concepts and direction.

Saleem N. Bhatti of UCL designed and implemented the log control function, including the managed object persistency, implemented the string-based CMIS filter manipulation language, designed and implemented the native version of the OSI Internet MIB-II, designed and implemented the public key based security mechanisms, designed the private key based ones and provided early feedback on OSIMIS design.

James Cowan of UCL implemented the first version of the object-oriented manager infrastructure, designed and implemented the generic MIB browser and designed and implemented the ingenious platform-independent GDMO compiler.

Thurain Tin, formerly of UCL, designed and implemented the second version of the object-oriented manager infrastructure, designed and implemented the Tcl-based version of the latter, contributed to the implementation of the access control function and implemented the generic CORBA to OSI-SM gateway.

Kevin McCarthy, formerly of UCL, designed and implemented the generic OSI-SM to SNMP gateway, contributed to the directory access concepts and integration, designed and implemented the private key based security mechanisms and contributed feedback to the OSIMIS design.

Costas Stathopoulos, formerly of ICS, Crete, Greece, designed and implemented the directory access for location transparency.

George Mykoniatis, formerly of NTUA, Greece, designed and implemented the intelligent monitoring facilities.

Jim Reilly, formerly of VTT, Finland, designed and implemented the first version of the metric monitor objects. He also designed and implemented the SMIC SNMP SMI to GDMO translator, based on an SNMP SMI compiler by David Perkins of Synoptics, USA.

Table of Contents

1. Introduction

1.1 The Telecommunications Management Network

The increasing sophistication of telecommunications infrastructures and, in particular, the introduction of broadband transmission and switching technologies require the use of sophisticated management facilities. These are important for harnessing the relevant multiplexing capabilities and for supporting the deployment of sophisticated telecommunications services with guaranteed Quality of Service (QoS) characteristics. The ITU-T has been developing the Telecommunications Management Network (TMN) [M3010] as the framework to support the *open* management of telecommunications networks and services. By management it is meant support for planning, provisioning, installing, operating, maintaining and administering these. The TMN corresponds to the *management* plane of the ITU-T Broadband Integrated Services Digital Network (B-ISDN) [BIDSN] reference model. It is an integral part for the operation of the network and services and complements the *control* and *user* B-ISDN planes.

The TMN is a logically separate network, overlaid over the telecommunications network being managed. It comprises a number of management applications, which are physically located "outside" the managed network. These communicate with each other and also access the network elements for the purpose of monitoring and controlling them. The management aspects of the network elements are considered as part of the TMN. A key aspect of the TMN model is its distributed hierarchical nature. Management applications are organised in layers of management responsibility, with each layer potentially comprising many distributed management applications. Higher layers build on the functionality provided by the underlying layers and offer increased encapsulation and abstraction. The TMN proposes the hierarchical separation of management responsibility into element, network, service and business management layers.

A key requirement for the TMN is *openness*. It should be possible for a network operator to buy network elements and management applications from various different suppliers and deploy them together into a TMN. This requires an agreed architecture with well-defined interfaces for managed elements and management applications. Additional key requirements include: the timely reaction to network events; the minimisation of load caused by management traffic; the graceful evolution to new generations of management functionality; the ability to cope with very large sizes of manageable entities; and the ability to cope with the problems caused by the wide geographical distribution of the relevant applications. In addition, good performance is desirable though the TMN does not have the same stringent real time requirements as the control plane functions.

The TMN started being developed by the ITU-T in the mid to late eighties and a key decision was to adopt emerging object-oriented concepts for management information modelling. Object-orientation has been a cultural achievement of information technology that supports better reusability, extensibility, genericity and system integrity [Rumb91]. The TMN needs a distributed object access technology as the basis of its interfaces which should also satisfy the key requirements identified above. The decision at the time was to adopt the emerging OSI Systems Management (OSI-SM) [X700] as the basis for the TMN information architecture and for the distributed object access across standardised interfaces.

OSI-SM was initially developed for the management of OSI data networks. It was the first OSI application to adopt a fully object-oriented approach. The OSI Directory [X500] was another OSI application that adopted some object-oriented principles but fell short of true object-orientation. In OSI-SM, clusters of managed objects model managed entities, being administered by applications in *agent* roles. Managed objects are specified in an abstract object-oriented information specification language, known as the Guidelines for the Definition of Managed Objects (GDMO) [X722]. Applications in *manager* roles access those objects in a distributed fashion in order to implement management policies. Distributed access is supported by the Common Management Information Service and Protocol (CMIS/P) [X710][X711], which was the first generic OSI protocol with "remote method call" semantics. A key aspect of the OSI-SM model is its support for multiple object access through sophisticated predicates and fine-grain control of notifications emitted by managed objects.

While TMN and OSI-SM are different, the former being the framework and the latter the supporting technology, they have typically been treated together, at least until recently. The development of both frameworks followed the ISO/ITU-T tradition, addressing the relevant functionality in an abstract fashion and disregarding software realisation aspects. The combined OSI-SM / TMN framework is mainly a *communications* concept, resulting in well-defined formats of interoperable messages but leaving unspecified aspects concerning the internal structure of applications.

The TMN framework [M3010] appears to be very complex. In addition, OSI-SM has been widely perceived as a technology difficult to implement, resource hungry and potentially resulting in poor performance [Rose91]. This is implicitly recognised in the TMN architecture which includes lightweight interfaces; these may be supported by "less capable" network elements. These lightweight interfaces need to be treated specially through mediation applications and this increases further the complexity of the overall framework.

In summary, the general perception of the TMN in the early nineties was as being too complex and ambitious, with its feasibility, implementability, performance and eventual deployment being widely doubted. In addition, despite the fact the TMN was supposed to be a large scale distributed system, there was no architectural provision for coping with the problems of distribution in terms of location transparency.

1.2 Thesis Motivation

The author developed an interest in OSI-SM as part of his work on the ESPRIT INCA[1] project in 1988. This involved the development of management applications for monitoring the activity of the OSI transport protocol, based on the emerging but incomplete OSI-SM standards [Knig89]. The author started following and contributing to the work of the British Standards Institution (BSI) on the protocol aspects of OSI-SM. The relevant standards reached a state of maturity in 1989 and the author implemented a complete version of CMIS/P under the ESPRIT PROOF project. When subsequently trying to implement management applications, it became clear that a generic approach was necessary in order to harness the complexity of the OSI-SM framework. The inherent object oriented aspects of GDMO could be exploited through mappings to object-oriented languages, hiding the underlying protocol aspects. The author subsequently designed and implemented a first version of a generic OSI-SM agent environment in early 1990 [Pav91a].

At the same time, the importance of the TMN was recognised in the RACE research framework, with four major projects addressing relevant technology aspects, NEMESYS, AIM, ADVANCE and GUIDELINE. The author started working in the NEMESYS project in late 1990 and became aware of a another framework and associated "culture", directly relevant to the realisation of distributed telecommunications software systems: the ISO/ITU-T Open Distributed Processing (ODP) [ODP]. ODP was influenced by the UK Alvey Advanced Networked Systems

[1] Details on the European projects the research work in this thesis was performed are given later, in section 1.4.

Architecture (ANSA) project [ANSA89a]. The latter realised early the need for an object-oriented distributed software framework instead of various application layer protocols. ODP and ANSA had been influential in the RACE research programme, including the NEMESYS project.

It was then that the author conceived the idea of a TMN object-oriented software platform: a development environment that would conform to the protocol aspects of the TMN framework for "on the wire" interoperability but would also adopt ODP concepts and provide easy to use Application Program Interfaces (APIs) for distributed object access. This environment should hide the complexity of the underlying OSI protocols but should provide access to the CMIS multiple object access and fine-grain event control features because of their importance for telecommunications management environments. In addition, the ODP access and location transparencies [X903] should be supported.

The relevant research work reused the author's experience on OSI upper layer protocols and applications and "married" it with ODP and ANSA concepts. The resulting TMN platform took a number of years to complete and is known as the OSI Management Information Service (OSIMIS) [Pav95b]. It was initially used in the NEMESYS project as the basis for first hierarchical TMN system with fully compliant TMN interfaces [Pav91b][Pav92b]. It was subsequently developed further and was used in a number of ESPRIT, RACE and ACTS projects. A substantial amount of research and development work took place in the RACE ICM project [ICM], between 1992 and 1995. It was then used to support one of the most complex TMN systems at the time [Grif96b][Reil96]. OSIMIS has also been used by numerous research institutions and companies, outside European research projects and has influenced a number of commercial developments. Finally, its software abstractions and APIs were input to standardisation work by the Network Management Forum (NMF).

Since the proposed TMN software platform was influenced by ODP concepts, an alternative approach would have been to consider the use of an ODP-based platform and add the necessary functionality for TMN environments. The author considered migrating the OSIMIS platform over ANSA in the first year of the ICM project. The relevant analysis, which is presented in this thesis, showed that this was not possible given a number of TMN requirements not met by ANSA. The Open Software Foundation's (OSF) Distributed Computing Environment (DCE) was considered as the second generation ODP-based platform. Examined carefully, this was a step sideways in comparison to ANSA and exhibited the same deficiencies when considered as the basis for TMN systems.

The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) can be seen as the third generation ODP-based platform. This is the first true object-oriented platform and presents a credible alternative to OSI-SM. Since its inception, a number of researchers have been considering its use for telecommunications management, given its generality and superior distribution aspects. Particularly important work has been the comparison of the OSI-SM and OMG object models [Rutt94] and the identification of generic rules for the translation between object specifications in the two frameworks [JIDM95]. The author proposes a solution which maintains the full OSI-SM functionality over CORBA, based on the experience gained by designing and implementing the OSI-SM based platform. A part of this solution was validated through implementation in the ACTS VITAL and REFORM projects.

Throughout the period of this research work, the author has been contributing to the ITU-T TMN architecture group. As a result of the relevant design and implementation experience, the author has proposed a number of simplifications to the TMN architectural framework [Pav96a] which are presented in this thesis.

1.3 Problem Statement and Approach

Given the motivation and history behind the research work in this thesis, the actual thesis statement can be formulated as follows:

The TMN is a powerful and relatively simple framework. The choice of OSI-SM as its base technology satisfies key requirements while problems of distribution can be addressed through the use of the OSI Directory. The TMN inherent object-oriented information specification aspects can be exploited through an object-oriented realisation model that conforms to the "spirit" of the ODP framework. OMG CORBA is the first ODP-influenced technology that satisfies key TMN requirements and can be used to replace seamlessly OSI-SM and OSI Directory as its base technology.

The OSI-SM and TMN frameworks are presented first. This presentation serves both as state-of-the-art but goes further by providing insights to a number of design decisions and associating them to the underlying requirements and fundamental assumptions. A number of modifications and extensions to the TMN framework are presented next. The modifications propose the simplification of the overall framework and rely on the fact that it is later shown that full scale OSI-SM/TMN technology is feasible, performant, easy and economical to provide. The extensions introduce location transparency and discovery facilities through the use of the OSI

Directory, which is chosen because of its federated nature. Location transparency is a very important property of distributed systems.

The key properties of object-oriented distributed frameworks are identified next. These cover ODP aspects such as access and location transparency, but a systematic presentation of the ODP framework is attempted later. The proposed OSI-SM/TMN realisation model and the various incarnations of ODP-based technologies, i.e. ANSA, OSF DCE and OMG CORBA, will be measured against those properties.

It is then shown in detail how the inherent object-oriented information specification aspects of the OSI-SM / TMN framework can be exploited through the use of object-oriented design and implementation methodology that results in an easy-to-use, distributed management platform. Object-oriented mappings for the CMIS service [X710], the Abstract Syntax Notation One (ASN.1) "network data typing" language [X208] and the Guidelines for the Definition of Managed Objects (GDMO) [X723] are presented. The programming language of choice is C++ [Strau86] but the presented concepts are general enough to be used in other object-oriented languages such as Smalltalk [Gold83] or Java [Sun96]. A client or manager mapping to the Tcl scripting language [Oust94] is also presented; this could be used for applications with Graphical User Interfaces (GUIs). A performance analysis and evaluation of the resulting platform shows that it has good performance characteristics and relatively modest resource requirements. Finally, the resulting platform is measured against the properties of object-oriented distributed frameworks and it is shown how it can support TMN applications. Given the fact that the ultimate validation of the proposed framework was achieved through the research and development of TMN systems, work based on the proposed platform is presented in Appendix A.

The ODP framework is presented next, along with related technologies such as ANSA, OSF DCE and OMG CORBA. This is more than a state of the art description, considering the ODP framework and relevant technologies in the context of telecommunications management and examining the potential advantages and open issues. The OSI-SM model is positioned in the ODP framework and it is explained why ANSA or the OSF DCE cannot be used as TMN base technologies. OMG CORBA is described in some detail, since it is the first ODP-based technology that satisfies almost all the key properties of distributed object frameworks. A solution is then presented for mapping OSI-SM onto OMG CORBA without losing any of its expressive power. This solution is based on the design and implementation experience gained from the OSI-SM based platform and considers the issues behind a native CORBA-based telecommunications management environment. Through this approach it is possible to move gradually towards a

CORBA-based solution, protecting existing investment in OSI-SM based technology. A relevant performance analysis examines the potential advantages and drawbacks compared to the OSI-SM approach.

The nature of the research work in this thesis can be thought as having three different facets:

- The modifications and extensions to the TMN architecture is research work of *architectural* nature. It relies on experience from the design, implementation, performance and resource requirements of real TMN systems. It also brings in ODP concepts such as location transparency. These modifications and extensions result in a simpler and more powerful framework.
- The design and development of the OSI-SM-based TMN platform is research work that targets the *validation* of the OSI-SM / TMN framework through *design* and *implementation*. It is though far from simple implementation work because of the requirement to adhere to the ODP vision of an easy-to-use, distributed object access framework with object-oriented APIs. When this research work started, it was not at all clear how CMIS, ASN.1 and GDMO could be mapped onto object-oriented software concepts. The work in this thesis has contributed in this direction.
- Finally, the research work that considers the use of CORBA as the base technology for future TMN systems is *specification* work which attempts to bring together the OSI-SM and CORBA / ODP frameworks. It relies on the understanding of the requirements and capabilities of the two frameworks, both theoretically and practically. A part of the resulting specification has been *validated* through design and implementation.

It should be mentioned that although a performance analysis is presented for both the OSI-SM and CORBA based frameworks, this is *not* a performance type of thesis. The aim of OSI-SM performance analysis is to validate the proposed framework from a performance perspective and to demonstrate that the proposed software architecture has relatively good performance characteristics. The aim of CORBA performance analysis is to compare and contrast it to OSI-SM, examine if the migration towards CORBA brings a performance advantage and assess issues of scalability. In summary, this research work is mostly architectural, specification and design work, validated through implementation.

This research work has been undertaken over a long period of time. As a result, it is inevitable that there exist TMN platform products today which have some of the characteristics of the software architecture presented in this thesis. This has been also aided by the fact that the

OSIMIS environment has been in the public domain since 1991 in order to promote the OSI-SM / TMN concepts. At least three commercial TMN platforms are known to be based on it and possibly many more. On the other hand, important findings regarding this research work have been published early i.e. [Pav91a], [Pav92a], [Pav93a], [Pav93b]. Products with similar functionality appeared in the market place around 1995.

The research work concerning the use of CORBA took place in the last two and a half years, after commercial CORBA implementations with C++ programming language bindings became available. While the research community started working on this issue since 1994, there is not a solution yet that reproduces the full OSI-SM functionality in a native CORBA environment. Most research has concentrated in solutions that support gateways between CORBA and OSI-SM environments. The solution proposed by the author targets native CORBA-based TMN applications throughout a TMN system i.e. a "CORBA to the switch" approach.

The research work in this thesis has being mainly based on the following state of the art work:

- the TMN [M3010] and OSI-SM [X700] architectural frameworks;
- the ODP [ODP] architectural framework;
- the ANSA software platform concepts [ANSA89a][ANSA89b];
- the software abstractions in the ISODE OSI upper layer development environment [ISODE][Rose90];
- the object-oriented design methodologies and principles [Booch91][Rumb91]; and
- the OMG CORBA framework [CORBA], the comparison of the OMG and OSI-SM object models [Rutt94] and the relevant object model mappings [JIDM95].

1.4 The Environment and Evolution of this Research Work

Most of the research work in this thesis has been undertaken in the context of applied research projects funded by the Commission of the European Union (CEU). A brief description of those projects and the evolution of this research work are presented in this section.

Early work on the validation of the evolving OSI-SM framework was initially undertaken in the Integrated Communications Networking Architecture (INCA) project, under the European Specific Research in Information Technology (ESPRIT) framework in 1988. The INCA project was addressing the validation of the emerging OSI upper layer protocols and applications. A design and implementation of the incomplete CMIS/P specifications was produced, together with management applications for monitoring the activity of the OSI transport layer. This work is described in [Knig89] and did not target generic, reusable software model. A procedural implementation paradigm was followed, using the C programming language [Kern78].

This work continued in the ESPRIT Primary Rate ISDN OSI Office Facilities (PROOF) project, which was investigating the issues of gateways between primary rate ISDN and IP or X.25 data networks[2]. A prototype gateway was to be produced in the project and an associated OSI-SM agent had to be designed and developed. It became then clear that a different, generic style for the realisation of OSI-SM agents was necessary, exploiting the object-oriented aspects of GDMO and achieving software reusability and extensibility. The first embryonic ideas on the potential generic structure of OSI-SM agents are described in [Knig90]. The author subsequently undertook research work towards the object oriented decomposition of OSI-SM agents and a first design and implementation was produced in early 1990. This research work is described in [Pav91a] targeting a generic, reusable, object-oriented agent infrastructure. The C++ programming language [Strau86] was used since the first relatively stable, bug-free compilers had become available at that time.

In late 1990, the author started working in the Network Management using Expert Systems (NEMESYS) project of the Research in Advanced Communications in Europe (RACE) framework. The latter was addressing issues in broadband communications and telecommunications services, with TMN being a central theme addressed by a number of other projects. NEMESYS was investigating the applicability of Advanced Information Processing

(AIP) technologies to TMN and the use of object-orientation, distributed systems and ODP principles were central to the project. It was then that the idea of a distributed TMN platform was conceived. The generic agent environment was developed further but it was complemented by generic manager infrastructure and generic applications, including a generic Management Information Base (MIB) browser [Pav92a]. The first hierarchical TMN system with fully compliant TMN interfaces was designed and implemented using the resulting TMN software platform; the functionality of this system, which addresses performance management aspects of Asynchronous Transfer Mode (ATM) networks, is described in [Pav91b][Pav92b]. Towards the end of 1991, this platform became available to companies and research institutions in the RACE programme under the name OSIMIS-2.95. The strange version number was supposed to mean "just before version 3.0", though it took another two years until version 3.0 was released!

The NEMESYS project had two successors in the RACE-II programme, the Integrated Communications Management (ICM) and Pre-Pilot in Advanced Resource Management (PREPARE) projects. The former was addressing ATM configuration and performance management while the latter was concentrating in inter-domain service management; both projects used the TMN architectural framework. The initial idea in ICM was to follow a true ODP approach, adopting either ANSA [ANSA89a] or DCE [DCE] as the platform. Support for location transparency through trading in ANSA and through name services in DCE was perceived as an important feature. On the other hand, mapping the OSI-SM model onto either of those technologies is problematic, for reasons explained in Chapter 4 of this thesis. As a result, OSIMIS was adopted and developed further in that project, including the addition of location transparency through the OSI Directory [X500]. In the PREPARE project OSIMIS was used together with another four commercial platforms (!), two of which were based on OSIMIS.

OSIMIS was developed much further in the course of the ICM project. Various features were added, including: object-oriented ASN.1 and GDMO compilers [Pav96b]; location transparency through the OSI Directory [Stath93][Stath95][Pav96a]; a full version of object-oriented manager infrastructure [Pav94b]; Tcl-based scripting manager infrastructure [Tin95][Pav96d]; intelligent monitoring objects [Pav96c]; generic gateways to the Internet Simple Network Management Protocol (SNMP) [Pav93c][McCar95]; meta-management facilities [Sartz95]; lightweight private key based security services including authentication, integrity and confidentiality services [Bhat96]; and object-based access control [Pav96b]. In parallel to the ICM project, the ESPRIT

[2] Such gateways between PR ISDN and IP are a commercial reality today, used a lot by Internet Service Providers (ISPs).

Management in a Distributed Application and Service Environment (MIDAS) contributed also significantly to the OSIMIS development, including a public key based version of the security services [Bhat95] which preceded the lightweight version mentioned above. The OSIMIS environment is described in [Pav93a], [Pav95a] and [Pav96b].

OSIMIS-3.0 was released in early 1993 [Pav93b] and OSIMIS-4.0 in early 1995 [Pav95b]. These have been the two official releases that have been used by numerous research institutions and influenced a number of commercial products. The OSIMIS-4.1 version contains the lightweight security mechanisms but it was never publicly released.

The RACE framework was followed by the Advanced Communications Technologies and Services (ACTS) one, which started towards the end of 1995. By that time, a significant amount of TMN research work had already been accomplished through the ICM and PREPARE projects and through a third important project that addressed service management issues using ODP, the Pan-European Reference Configuration for IBC Service Management (PRISM) project. The ICM, PREPARE and PRISM projects have published their research results in books, [ICM], [PREPARE] and [PRISM] respectively. Given this culmination of the TMN research work in RACE, ACTS concentrated in research on integrated management and service control frameworks adhering to the emerging Telecommunications Information Networking Architecture (TINA) [TINA]. The author has been involved in the VITAL and REFORM projects.

The Validation of Integrated Telecommunications Architecture for the Long-term (VITAL) project attempts to validate the TINA framework. This is done through the design and implementation of an integrated architecture comprising both multimedia service control and ATM network management aspects. On the other hand, The Resource and Fault Restoration and Management (REFORM) project can be thought as the continuation of the ICM project. It adds fault management to the ATM configuration and performance management functions while it uses the TINA instead of the TMN architectural framework.

Since the base technology for TINA systems is a CORBA-based platform known as the Distributed Processing Environment (DPE), an important issue is how the TMN methodologies, principles, specifications and existing applications will be retained and reused. The key issue is how to map the OSI-SM / Directory model to CORBA; this issue has attracted significant attention from the research community. Relevant research work was undertaken by the author in the context of the VITAL and REFORM projects, resulting in a model that retains the OSI-SM power and expressiveness. This model has been validated through implementation [Pav97b][Pav97d] and is presented in Chapter 4 of this thesis. There have been two different implementations of this model: a gateway one, which allows CORBA client or manager objects to access OSI-SM agents; and a native one, in which TMN applications have native CORBA-based interfaces. These implementations were produced in the course of 1996 and 1997 respectively, using a combination of the OSIMIS infrastructure and a commercial CORBA platform.

Figure 1-1 shows the various research projects between 1989 and 1997. It should be noted that the VITAL and REFORM projects are still active. The various OSIMIS versions are also shown. The last two versions which include the CORBA-based gateway and native CORBA agents have not been released outside the ACTS research community.

It should be finally stated that OSIMIS is not only the work of the author. A number of researchers, at UCL and elsewhere, have contributed to it. Given the fact that this research work has been undertaken in the context of collaborative research projects, the role of other researchers to the contributions presented is made clear throughout the thesis. The large majority of the work presented though constitutes research work by the author.

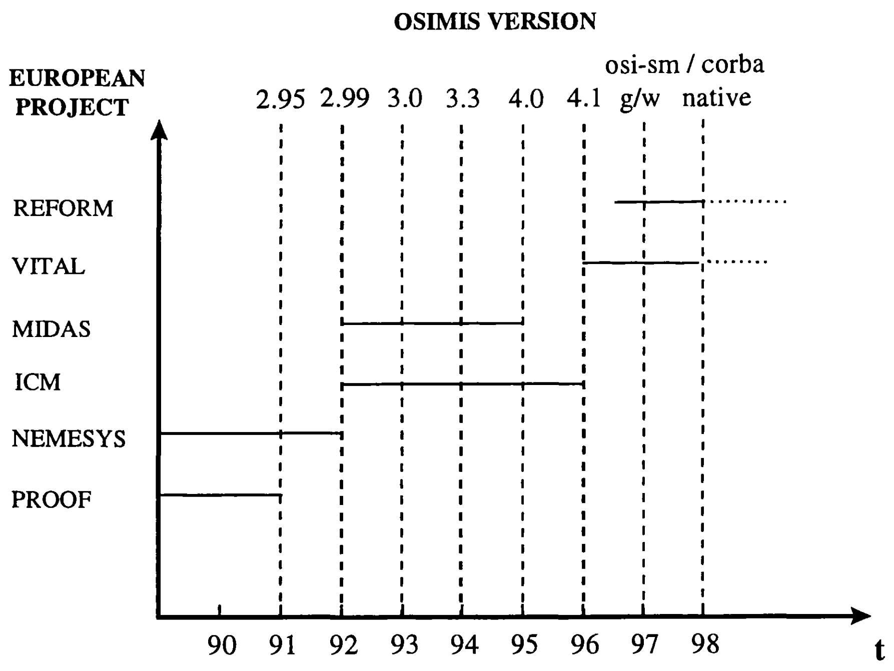


Figure 1-1 European Research Projects and OSIMIS Evolution

1.5 The Difficult Path to the Achievements

At the end of 1997 TMN systems have started to become a reality, especially in the areas of broadband transmission, switching and mobile network environments. The initial vision of the TMN has been translated into real systems eight to nine years later. The research work presented in this thesis has contributed to this direction. The road to this point though has not been easy. Initially, the overall framework was considered too complicated, using abstract object-oriented concepts with no concrete counterparts, unimplementable, resulting in poor performance and requiring excessive computing resources. Proponents of the Internet management approach [SNMP] argued that reliable connection-oriented transport is not suitable for management and object-orientation is an unnecessary complication. ODP proponents argued that the CMIS access model is complicated, difficult to implement and unnecessary. The fact that OSI protocols were used was also seen as an additional negative point. This negative perception was also exacerbated by the fact that early implementations of OSI upper layer protocols had been bad, resulting in cumbersome applications and poor performance.

Three real incidents are described below which demonstrate the general perception of OSI-SM and TMN in the first half of the nineties,. They also try to demonstrate the pioneering role of this research work in demystifying the OSI-SM / TMN framework and establishing its true properties and qualities.

Incident 1: January 1993

In early 1993 a new researcher joined the UCL network and service management group to work in one of the projects. He had previously worked for one of the commercial suppliers of OSI infrastructure and had participated in the development of their OSI-SM product. Since he was going to work with OSIMIS, the author gave to him a small demonstration that involved the manipulation of management information through a number of command line tools and also through a graphical MIB browser. The relevant agent and manager applications were running in different computers on the local network, using full TMN Q_3 protocol stacks over the Internet TCP/IP protocols [Q3], as described in Chapter 3.

That researcher asked a number of questions before the short demonstration but remained silent in the course of it. When the author asked him about his thoughts, he faced an expression of mixed suspiciousness and amazement, being asked back: "*Are you claiming these applications have full OSI-SM functionality and access each other in a fraction of a second?*". A second quick demonstration with the timing flag on for one of the programs revealed around 150 ms for establishing an association, and around 50ms for retrieving the root MIB object and its attributes, based on the computing power available at the time (detailed performance measurements are presented in section 3.7 of Chapter 3). He subsequently mentioned that the products he had encountered needed around 15 seconds (!) for establishing an association and more than one second for retrieving an object under similar conditions. A similar figure for association establishment is also mentioned in [Rose90].

Incident 2: April 1993

The first paper on generic OSI-SM agent infrastructures [Pav91a] and the subsequent early releases of OSIMIS had generated a lot of interest in the UCL work on OSI-SM and TMN. UCL was asked to exhibit OSIMIS in the 3rd IFIP/IEEE Integrated Management Symposium [IM-III] in San Francisco, during April 1993. The relevant costs were levied so that the exhibition could comprise a "product" that was more than a CMIS/P protocol stack, with real OSI-SM agents and generic managers. Since the author was also going to present a tutorial on "Implementing OSI-SM" [Pav93a], a demonstration was prepared. This involved the demonstration of a number of aspects of the OSI-SM model, including event management [X734], log control [X735], MIB browsing etc. The management interactions were shown both over the local network and also over the Internet, accessing agents at UCL. The latter supported the OSI version of TCP/IP MIB [Laba91b].

This demonstration proved to be immensely popular, since it provided a tangible proof that OSI-SM was after all both implementable and performant, in contrast to all the verbal attacks by the SNMP community during the conference. The OSI-SM booth was situated just next to the SNMP one, which included a number of vendors. This gave to the author the following idea: he "invited" SNMP vendors to conduct comparative tests by accessing SNMP agents at UCL over the Internet while the author would also conduct tests by accessing the equivalent OSI-SM agent at UCL through CMIS/P. This would allow to assess both success in retrieving complex information, such as routing tables, and also to compare relative performance. While the numerous visitors of the two booths got really excited, the SNMP vendors declined repeatedly the "invitation". The author finally tried the experiment himself by using the generic SNMP manager that comes with the ISODE package [ISODE]. The result was that the OSI-SM based experiment succeeded every time, exhibiting much better performance characteristics due to intelligent remote object access through "scoping and filtering".

Incident 3: September 1995

European research projects are typically audited every year. The final audit for the ICM project took place in September 1995 and included a demonstration after the project's presentation and questions by the auditors. The demonstration comprised the ICM TMN system, described in detail in [Gri96b] and [ICM], which provided ATM Virtual Path Connection and Routing Management (VPCRM) services and also ATM-based Virtual Private Network (VPN) services. This system was the most complex real TMN built at the time (and maybe to date), comprising twelve different types of TMN Operations Systems (OSs). Instances of those existed in different domains, providing end-to-end VPN services and intra-domain VPCRM services. The system had operated over real networks, but for the purpose of the demonstration it was operating over a simulated network of eight nodes. The processing power hungry simulator was running on one workstation, posing as a set of ATM switches with individual TMN Q_3 interfaces, while the operations systems were running on another workstation.

The demonstration was given in parallel to a slide presentation, which was explaining the various features of the system. The auditors could see those features and relate them to action on the screens of the two workstations. While the demonstration was going on, one of the auditors asked the author: "*Are you sure all the OSs in your TMN system have full Q_3 interfaces and do not simply interwork through an ad-hoc lightweight mechanism?*" After the affirmative reply, a second question followed: "*Can you then please show the trace of CMIS/P messages these applications send and receive?*". We had to stop the system and re-start it, configuring relevant output and redirecting it to log files so that we could observe those. The expression on the auditors' faces was little short of amazement since they had totally different ideas about the overhead of a fully compliant TMN OS.

In summary, TMN suffered initially from bad design and engineering. The fact early versions of OSIMIS were made publicly available early enough helped a new generation of better TMN products. Despite the relevant progress though, it is still non uncommon to see requirements for workstations with at least 96 Mb of memory and very powerful CPUs in order to run a TMN OS or do TMN system development. This is in contrast to the relatively lightweight OSIMIS approach, which is acknowledged in [Deri97] when comparing various management technologies and platforms.

1.6 Overview and Style of this Thesis

This thesis is organised in the following fashion. This chapter was the introduction to the thesis while Chapter 5 presents the summary and conclusions. Chapters 2, 3 and 4 constitute the core material of the thesis and can be thought as "super chapters", each addressing a set of related topics. These topics are presented in the first level sections of those chapters and can be thought as "chapters within a chapter". State of the art work is presented throughout this thesis, either in separate sub-sections or just before the author's own work on a particular subject. For example, research work on CMIS APIs is presented just before the author's own work on the subject.

Chapter 2 addresses the OSI-SM / TMN architectural aspects. It starts with two sub chapters on OSI-SM and TMN respectively, which serve mainly as of state of the art description but they also provide insights to a number of design decisions. The third sub-chapter proposes a number of modifications and extensions to the TMN model and architecture, supported by a relevant analysis. This constitutes the first research contribution of the thesis.

Chapter 3 addresses the realisation of the simplified and extended OSI-SM / TMN framework in an object-oriented fashion. The target to provide an easy-to-use distributed management platform that exhibits characteristics of object-oriented distributed processing frameworks. This chapter comprises a number of sub-chapters as follows. An introduction to object-oriented distributed systems is presented first, identifying their key properties. Issues behind realising the CMIS/P protocol are presented next, examining potential policies for the relevant API. Alternative lightweight mappings that avoid the use of a full OSI stack are also considered. The next sub-chapter examines issues behind treating ASN.1 data types in an object-oriented fashion. This leads to the next two sub-chapters that examine object-oriented mappings for managing and managed objects respectively. The issues behind synchronous and asynchronous remote execution models are then presented. A performance analysis and evaluation of the overall framework follows. Finally, the proposed framework is further validated against the desired properties of object-oriented distributed systems and the needs of TMN applications as identified in Chapter 2.

Chapter 4 considers the mapping of the OSI-SM / TMN model onto emerging distributed object frameworks that follow ODP principles, using OMG CORBA as the target framework and technology. An introduction to ODP is presented first. This serves as state of the art description but also positions OSI-SM in the ODP framework. Different incarnations of ODP-based technologies are presented next. These include ANSA, the OSF DCE and OMG CORBA, the latter considered in more detail. This is more than a state of the art description which positions those technologies in the context of telecommunications management and examines their characteristics against the identified properties of object-oriented distributed systems. The next sub-chapter examines in detail the issues behind using OMG CORBA instead of OSI-SM in TMN environments and proposes a solution that retains the expressive power of the latter. Finally, a performance analysis and evaluation of the CORBA-based framework is presented.

Chapter 5 brings together the various strands of the thesis and summarises the main findings. It explains the significance of this thesis and also identifies areas in which work could be developed further. A number of appendices follow at the end, together with the acronyms and bibliography.

It is also worth mentioning related issues that are *not* discussed in this thesis. Intrusive and inter-domain management is impossible without security. This thesis does not address security issues, though the author has been involved in relevant research work [Bhat96][Pav96b]. Another management paradigm is that of moving code rather than data between management applications. This was first proposed in [Yemi91] and is known as "management by delegation". The advent of languages such as Java [Sun96] and GUI technologies such as the World Wide Web (WWW) open new possibilities for management using mobile code. These will certainly have an impact on telecommunications management architectures in the long term but they are not examined in this thesis. Finally, another approach to management that has evolved over the last years is domain and policy based management [Slom89][X749]. This is somehow orthogonal to the work presented in this thesis and is not addressed here.

Considering the style of this thesis, a number of C++ object class specifications and code extracts are presented in Chapter 3 in order to demonstrate the various features of the proposed software infrastructure. These are fundamental since they demonstrate the relevant properties and user-friendliness, so they could have *not* been moved to appendices. In a similar fashion, CORBA design aspects are demonstrated through the use of specifications in the OMG Interface Definition Language (IDL) in Chapter 4. The Object Modelling Technique (OMT) [Rumb91] together with objects instance diagrams are used to demonstrate aspects of object-oriented design.

Finally, as the reader may have already noticed, "we" or "our" is used instead of "I" or "mine" when describing research work by the author throughout this thesis [3].

[3] This has nothing to do with the "royal *we*" but simply reflects the author's writing style.

2. OSI Systems Management and the Telecommunications Management Network

2.1 Introduction

Chapter 2 of this thesis introduces first OSI Systems Management (OSI-SM) [X700][X701] and the Telecommunications Management Network (TMN) [M3010]. It then proposes a number of modifications and extensions to the TMN architectural framework which aim to enhance and simplify it considerably, without sacrificing any important aspects for open interoperable telecommunications management. These modifications and extensions are supported by relevant analysis in this chapter and are also backed up by experimental results presented in Chapter 3.

OSI-SM is currently the base technology for the TMN, which provides the architectural framework for telecommunications network and service management. A relevant introduction is important since the thesis proposes a novel approach for the realisation of OSI-SM/TMN systems through distributed object-oriented software platforms. This makes a detailed understanding of both OSI-SM and TMN necessary for following the rest of the thesis. While this presentation is based on the relevant standards, it tries to shed light on a number of issues, explaining the reasons behind the various architectural choices and associating them to the underlying requirements. This differentiates it from other presentations in the literature and, as such, it constitutes to some extent a research contribution in its own right. The presentation of the TMN framework in particular extends beyond the relevant standards or other presentations in the literature and presents a number of clarifications in a way that has never been attempted before.

The last part of this chapter proposes a number of extensions and modifications to the TMN model and architecture. These include the introduction of the OSI Directory [X500] to support distribution and discovery services and the simplification of the framework through the removal of the Q_x and F interfaces. While the former emanates from the requirements of the TMN as a large scale distributed system, the latter relates to the fact that Chapter 3 of the thesis shows that full scale OSI-SM/TMN technology based on Q_3 interfaces is feasible, performant and relatively economical to provide in terms of both required resources and development time. This obviates the need for lightweight interfaces which necessitate special support in order to be able to cope with the resulting heterogeneity. The architectural revision of the TMN and the clarification of a number of related issues constitute the key research contribution of this chapter.

This chapter is organised as a “super-chapter” in a similar fashion to chapters 3 and 4 of this thesis, in the sense that sections 2.2, 2.3 and 2.4 can be considered as “sub-chapters” within a chapter.

Section 2.2 presents OSI-SM model. It starts by summarising relevant work in the literature and discusses management requirements in terms of the management functional areas (section 2.2.1). It then presents the manager-agent model (section 2.2.2), which is subsequently elaborated in terms of the information modelling aspects (section 2.2.3) and information access aspects (section 2.2.4). Generic management functionality emanating from the management functional areas is finally presented in section 2.2.1.

Section 2.3 presents the TMN framework and principles. It starts by summarising relevant work in the literature and positioning the TMN in the Broadband ISDN context (section 2.3.1). It then presents various aspects of the TMN architecture (section 2.3.2), including the functional and physical architectures (sections 2.3.2.1 and 2.3.2.2), the logical layering aspects (section 2.3.2.3), the interface specification methodology (section 2.3.2.4) and the decomposition of TMN applications into constituent components (section 2.3.2.5).

Section 2.4 presents the modifications and extensions to the TMN model and architecture. It first explains the issues behind providing distribution and discovery services using the OSI Directory (section 2.4.1). It then discusses issues on adaptation and mediation and proposes the removal of the Q_x interface (section 2.4.2). It subsequently discusses issues on the F interface and proposes its removal (section 2.4.3). The TMN architecture is then revisited and simplified (section 2.4.4). The section closes with a final discussion on the TMN framework.

Finally, section 2.5 highlights the research contributions in this chapter and paves the way to Chapter 3.

2.2 OSI Systems Management

In this section we examine the ISO/ITU-T OSI Systems Management (OSI-SM) [X701] approach to network management which is currently the base technology for the TMN. This is an overview that explains the reasons for the various architectural decisions and associates them to the relevant requirements. A basic understanding of data network principles and in particular of the 7 layer OSI Reference Model (OSI-RM) [X200] is assumed. A good general introduction can be found in [Tanen96].

OSI-SM was the last set of OSI application layer standards to be addressed and, as such, it received considerable attention. The relevant standardisation effort lasted for almost a decade and it was the first OSI application to adopt fully object-oriented information specification and access principles. The OSI Directory [X500] was another OSI application that had adopted similar concepts but it fell short of true object-orientation. A number of tutorial and survey papers in the literature and also books have addressed OSI-SM, these are briefly summarised below.

[Kler88], [Jeff88], [Slum89], [Coll89] and [Smith90] all provide early descriptions of the OSI-SM framework and the developing standards at the time. [Kler93] and [Yemi93] provide better overviews since the relevant standards were by that time fairly mature, the first concentrating on information modelling aspects and the second addressing the framework as a whole. A number of books have also addressed OSI-SM. Among those, [Jeff92] addresses the subject best in the author's opinion. Three of the authors of that book have also contributed chapters to [Slom94]. [Lang94] covers the model and standards, [Tuck94] concentrates in the structure of management information and [Jeff94] covers the Guidelines for the Definition of Managed Objects. [Stal94] is another book that addresses partly the subject but it is brief in its description, concentrating mainly on the Simple Network Management Protocol (SNMP) [SNMP].

The material in this section is based to a large extent on [Pav97a], a chapter in [Aida97] which presents a comparative study of OSI-SM, Internet SNMP and ODP / OMG CORBA as technologies for telecommunications network management.

2.2.1 Management Functional Areas

OSI Systems Management standardisation followed a top-down approach, with a number of functional areas identified first. The reason for identifying those was not to describe exhaustively all relevant types of management activity but rather to investigate their key requirements and to address those through generic management infrastructure. The identified areas were *Fault*, *Configuration*, *Accounting*, *Performance* and *Security Management* [X700] and are collectively referred to as *FCAPS* from their initials. Their generic requirements are supported by the Systems Management Functions (SMF) [SMF]. The same functional areas have also being adopted by the TMN. We present here the typical activities in each functional area, identify their generic requirements and list the relevant SMFs.

Fault Management addresses the generation of error specific notifications (*alarms*), the logging of error notifications at source and the testing of network resources in order to trace and identify faults. Fault management systems should undertake alarm surveillance activities (analysis, filtering and correlation), perform resource testing and provide fault localisation and correction functions. The key requirements are event-based operation, a well-defined generic set of alarms and a testing framework. The relevant SMFs are event reporting [X734], logging [X735], alarm reporting [X733] and testing [X737][X745].

Configuration Management deals with initialisation, installation and provisioning activities. It allows the collection of configuration and status information on demand, provides inventory facilities and supports the announcement of configuration changes through relevant notifications. The key requirements are event-based operation, control of change, generic resource state and relationship representation, scheduling, time management, software distribution and system discovery facilities. The relevant SMFs are event reporting [X734], logging [X735], object [X730], state [X731] and relationship [X.732] management, scheduling [X746], time management [X743], software distribution [X744] and shared management knowledge [X750].

Accounting Management deals with the collection of accounting information and its processing for charging and billing purposes. It should enable accounting limits to be set and costs to be combined when multiple resources are used in the context of a service. The key requirements are event based operation, in particular logging, and a generic usage metering framework. The relevant SMFs are event reporting [X734], logging [X735] and accounting metering [X742].

Performance Management addresses the availability of information in order to determine network and system load under both natural and artificial conditions. It also supports the collection of performance information periodically in order to provide statistics and allow for capacity planning activities. Performance management needs access to a large quantity of network information and an important issue is to provide the latter with a minimum impact on the managed network. Key requirements are the ability to convert raw traffic information to traffic rates with thresholds and tidemarks applied to them; the periodic summarisation of a variety of performance information for trend identification and capacity planning; the periodic scheduling of information collection; and the ability to determine the response time between network nodes. The relevant SMFs are event reporting [X734], logging [X735], metric monitoring [X739], summarisation [X738], scheduling [X746] and response time monitoring [X748].

Security Management is concerned with two aspects of systems security. The *management of security*, which requires the ability to monitor and control the availability of security facilities and to report security threats or breaches. And the *security of management,* which requires the ability to authenticate management users and applications, to guarantee the confidentiality and integrity of management exchanges and to prevent unauthorised access to management information. Authentication, integrity and confidentiality services are common to all OSI applications and are addressed for the whole of the OSI framework in [X800][GULS]. The key requirements in OSI management are support for security alarms, facilities for security audit trail and the provision of access control. The relevant SMFs are event reporting [X734], logging [X735], security alarm reporting [X736], security audit trail [X737] and access control [X741].

A common requirement in all the functional areas is event-driven management through event reporting and logging facilities. The systems management functions [SMF] are described in section 2.2.5.

2.2.2 The Manager-Agent Model

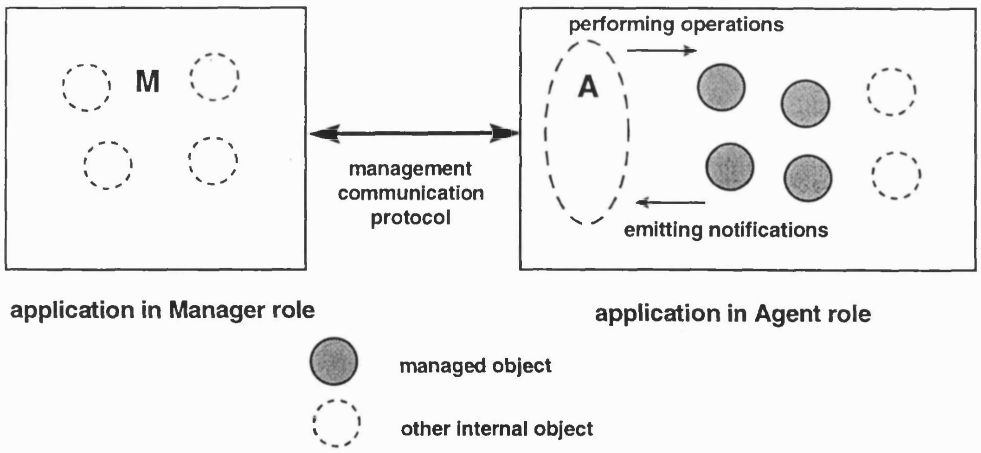

M
A
performing operations
emitting notifications
management communication protocol
application in Manager role
application in Agent role
managed object
other internal object

Figure 2-1 The Manager-Agent Model

OSI management has introduced the manager-agent model. A simplified version of this model has also been adopted by the Internet SNMP [SNMP]. According to the model, manageable resources are modelled by managed objects at different levels of abstraction. Managed objects encapsulate the underlying resource and offer an abstract access interface at the object boundary. The management aspects of entities such as network elements and distributed applications are modelled through "clusters" of managed objects, seen collectively across a management interface. The latter is defined through the formal specification of the relevant managed object types or classes and the associated access mechanism, i.e. the management access service and supporting protocol stack. Management interfaces can be thought as "exported" by applications in agent roles and "imported" by applications in manager roles. Manager applications access managed objects across interfaces in order to implement management policies. Distribution aspects are orthogonal to management interactions and are supported by other means.

OSI management is primarily a communications framework. Standardisation affects the way in which management information is modelled and carried across systems, leaving deliberately unspecified aspects of their internal structure. The manager-agent model is shown in Figure 2-1. Note that manager and agent applications contain other internal objects that support the implementation of relevant functionality. These are not visible externally, so they are depicted with dotted lines.

The management access service and protocol carries the parameters of operations to managed objects and returns relevant results, so it can be loosely described as a "remote method execution" protocol (in object-oriented systems, an object's procedure is called a "method"). The relevant parameters and results are a superset of those available at the object boundary, allowing to de-reference an object by name and to select a number of objects to perform an operation. In fact, the agent offers an object-oriented database-like facility which has the effect that *one* operation across the network may result in *many* operations to managed objects inside the agent, with a "consolidated" result passed back. In the other direction, notifications emitted by managed objects are discriminated internally within the agent, based on criteria preset by manager applications. This mechanism eliminates unwanted notifications at source and forwards useful notifications *directly* to interested managers.

The above facilities result in less management traffic and increase the timeliness of retrieved management information. These are key requirements in most management environments and, in particular, architectural requirements for the TMN as it will be described in section 2.3.2. In fact, the manager-agent model was designed for the purpose of supporting such facilities. Briefly positioning this model in the Open Distributed Processing (ODP) [ODP] framework that will be described in Chapter 4, an OSI agent acts as a naming server, trader, notification server and object access server with respect to the managed objects it administers. All these facilities are tightly coupled with the associated managed objects within the same network node. Chapter 4 explores relevant relationships with ODP in more detail and explains how this model can be transposed onto the ODP framework.

The manager and agent roles are not fixed and management applications may act in both roles. This is the case in hierarchical management architectures such as the TMN [M3010]. In hierarchical management models, managed objects exist also in the agent aspect of management applications, offering views of the managed network, services and applications at higher levels of abstraction. Management functionality may be organised in different layers of management responsibility: element, network, service and business management according to the TMN model. Management applications may act in dual manager-agent roles, in either peer-to-peer or hierarchical relationships. Figure 2-2 shows three types of management organisation: centralised, flat and hierarchical. The hierarchical model is best exemplified by TMN which uses OSI-SM as its base technology. Note that in both flat and hierarchical models, management applications are hybrid, assuming both the manager and agent roles.

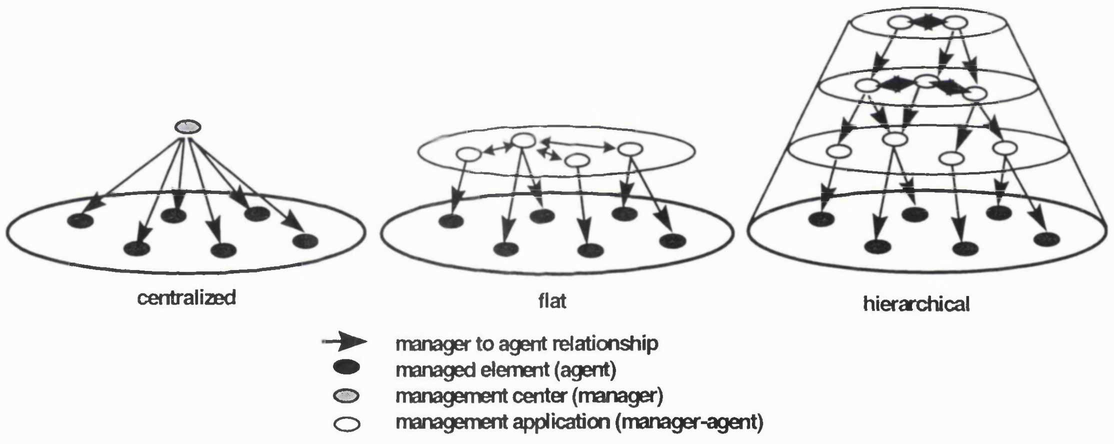

centralized
flat
hierarchical
manager to agent relationship
managed element (agent)
management center (manager)
management application (manager-agent)

Figure 2-2 Models of Management Organisation

2.2.3 The Management Information Model

The OSI-SM Management Information Model (MIM) is defined in [X720] and uses object-oriented principles. A systematic introduction to object-oriented systems is given in section 3.2.1 of Chapter 3. A reader with no relevant exposure is advised to read that section first, since relevant terms and concepts are used in this section.

An OSI Management Information Base (MIB) defines a set of Managed Object Classes (MOCs) and a schema which defines the possible containment relationships between instances of those classes. There may be many types of relationships between classes and their instances but containment is treated as a primary relationship and is used to yield unique names. The smallest re-usable entity of management specification is not the object class, as is the case in other O-O frameworks, but the *package*. Object classes are characterised by one or more mandatory packages while they may also comprise conditional ones. An instance of a class must always contain the mandatory packages while it may or may not contain conditional ones. The latter depends upon conditions defined in the class specification. Managing functions may request that particular conditional packages are present when they create a managed object instance.

A package is a collection of attributes, actions, notifications and associated behaviour. Attributes have associated syntaxes specified in ASN.1 [X208] which may be of arbitrary complexity. A number of generic attribute types have been defined in [X721], namely *counter*, *gauge*, *threshold* and *tidemark*; resource-specific types may be derived from these. The fact that attributes may be of arbitrary syntax provides useful flexibility, albeit at the cost of additional complexity. For example, it allows the definition of complex attributes such as threshold, whose syntax include fields to indicate whether the threshold is currently active or not and its current comparison value.

OSI managed object classes and packages may have associated actions that accept arguments and return results. Arbitrary ASN.1 syntaxes may be used for the arguments and results, in a similar fashion to attributes, providing a fully flexible "remote method" execution paradigm. Exceptions with MOC-defined error information may be emitted as a result of an action. The same is also possible as a result of operations to attributes under conditions that signify an error for which special information should be generated. Object classes and packages may also have associated notifications, specifying the condition under which they are emitted and their syntax. The latter may be again of an arbitrary ASN.1 type. By behaviour, one means the semantics of classes, packages, attributes, actions and notifications and the way they relate as well as their relationship to the entity modelled by the class.

OSI Management follows a fully O-O paradigm and makes use of concepts such as inheritance. Managed object classes may be specialised through subclasses that inherit and extend the characteristics of superclasses. The use of inheritance and packages allows re-usability and extensibility of specifications. It may also result in software reusability if an object-oriented design and development methodology is used, as explained in Chapter 3. As an example of inheritance, a transport protocol entity class (tpProtocolEntity) may inherit from an abstract protocol entity class (protocolEntity) which models generic properties of protocol entities, e.g. the operational state, the service access point through which services can be accessed, etc. By abstract class it is meant a class that is never instantiated but serves inheritance purposes only. In the same fashion, an abstract connection class may model generic properties of connection-type entities such as the local and remote service access points, the connection state, emit creation and deletion notifications, etc. The inheritance hierarchy for those classes is shown in the left part of Figure 2-3, using the Object Modelling Technique (OMT) notation [Rumb91].

It should be noted that conditional packages allow for dynamic (i.e. run-time) specialisation of an object instance while inheritance allows only for static (i.e. compile-time) specialisation through new classes. As pointed out in [Tuck94], in order to achieve the same effect with inheritance instead of packages, N conditional packages would necessitate the definition of 2^N additional classes! Any requirements for further subclassing would make the combinatorial explosion even worse.

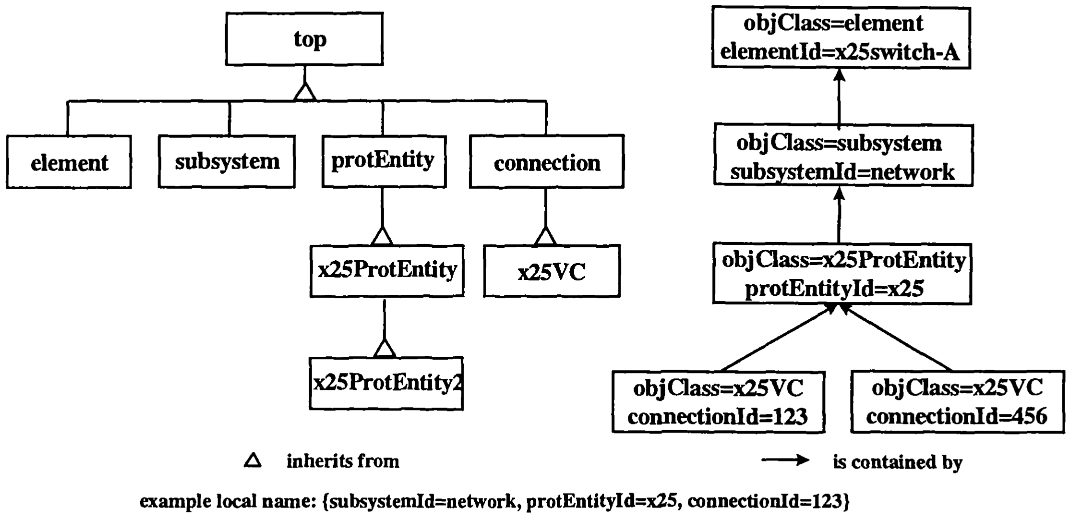

top
element
subsystem
protEntity
connection
x25ProtEntity
x25VC
x25ProtEntity2
objClass=element elementId=x25switch-A
objClass=subsystem subsystemId=network
objClass=x25ProtEntity protEntityId=x25
objClass=x25VC connectionId=123
objClass=x25VC connectionId=456
inherits from
is contained by
example local name: {subsystemId=network, protEntityId=x25, connectionId=123}

Figure 2-3 Example OSI Inheritance and Containment Hierarchies

The specification of manageable entities through generic classes which are used only for inheritance and re-usability purposes may result in generic managing functions in manager applications through polymorphism across a management interface. For example, it is possible to provide a generic connection-monitor application that is developed with the knowledge of the generic connection class only. This may monitor connections in different contexts, e.g. X.25, ATM, etc. disregarding the specialisation of a particular context. That way, reusability is extended to managing functions as well as managed object classes and their implementations.

In OSI management, a derived class may extend a parent class through the addition of new attributes, actions and notifications; through the extension or restriction of the value ranges; and through the addition of arguments to actions and notifications. Multiple inheritance is also allowed and it has been used extensively by information model designers in standards bodies. Despite the elegant modelling that is possible through multiple inheritance, such models cannot be easily mapped onto relevant facilities in O-O programming environments (e.g. C++ multiple inheritance) as discussed in Chapter 3. Multiple inheritance is a powerful O-O specification technique but increases complexity.

A particularly important aspect behind the use of object-oriented specification principles in OSI management is that they may result in the allomorphic behaviour of object instances. *Allomorphism* is similar to polymorphism but has the inverse effect: in polymorphism, a managing function knows the semantics of a parent class in an inheritance branch and performs an operation on an instance which responds as the leaf class. In allomorphism, that instance should respond as the parent class, hiding completely the fact it belongs to the leaf class. For example, a polymorphic connection monitor application can be programmed to know the semantics of the connection class and only the syntax of specific derived classes through meta-data. When it sends a "read all the attributes" message to a specific connection object instance, e.g. x25Vc, atmVcc, etc., it wants to retrieve all the attributes of that instance, despite the fact it does not understand the semantics of the specific "leaf" attributes. In allomorphism, a managing system programmed to know a x25ProtocolEntity class should be able to manage instances of a derived x25ProtocolEntity2 class without knowing of this extension. In this case, operations should be performed to the x25ProtocolEntity2 instance as if it were an instance of the parent x25ProtocolEntity class, since the derived class may have changed the ranges of values, added new attributes, arguments to actions, etc.

Polymorphism is a property automatically supported by O-O programming environments while allomorphism is not and has to be explicitly supported by management infrastructures as explained in Chapter 3. Allomorphic behaviour may be enforced by sending a message to an object instance and passing to it the object class as an additional parameter, essentially requesting the object to behave as if it were an instance of that class. When no class is made available at the object boundary, the instance behaves as the actual class i.e. the leaf class in the inheritance branch. Allomorphic behaviour is very important since it allows the controlled migration of management systems to newer versions by extensions of the relevant object models through inheritance, while still maintaining compatibility with previous versions. This is particularly important in management environments since requirements and understanding of the problem space are expected to be continuously evolving. Allomorphism hides extensions at the agent end of the manager-agent model. Extensions in managing systems should be hidden by programming them in advance to be able to revert to the "base" information model if this is what it is supported across a management interface. Though possible, this requires additional effort and increases complexity.

The root of the OSI inheritance hierarchy is the top class which contains attributes self-describing an object instance. These attributes are the objectClass, whose value is the actual or leaf-most class; packages, which contains the list of the conditional packages present in that instance; allomorphs, which contains a list of classes the instance may behave as; and nameBinding, which shows where this instance is in the naming tree as explained next. For example, in the instance of the x25ProtocolEntity2 class mentioned before, objectClass would have the value x25ProtocolEntity2 and allomorphs would have the value { x25ProtocolEntity }[1]. When an instance is created by a managing function, the conditional packages may be requested to be present by initialising accordingly the value of the packages attribute, which has "set by create" properties.

Managed object classes and all their aspects such as packages, attributes, actions, notifications, exception parameters and behaviour are formally specified in a notation known as Guidelines for the Definition of Managed Objects (GDMO) [X722]. GDMO is a formal object-oriented information specification language which consists of a set of templates. A "piecemeal" approach is followed, with separate templates used for the different aspects of an object class i.e. class, package, attribute, action, notification, parameter and behaviour templates. GDMO specifies formally only syntactic aspects of managed object classes. Semantic aspects, i.e. the contents of behaviour templates, are expressed in natural language. The formalisation of managed object behaviour has been a research topic that attracted considerable attention [Fest93] [Fest95] [Katch95]. Recently, the use of formal specification techniques such as System Definition Language [Z100] and Z [Spiv89] are considered by the ITU-T in order to reduce the possibility of ambiguities and misinterpretations and to increase the degree of code automation.

OSI-SM managed object instances are named through a mechanism borrowed from the OSI Directory [X500]. Managed object classes have many relationships but containment is treated as a primary relationship that yields unique names. Instances of managed object classes can be thought as logically containing other instances. As such, the full set of managed object instances available across a management interface is organised in a Management Information Tree (MIT). This requires that an attribute of each instance serves as the "naming attribute". The tuple of the attribute and its value form a Relative Distinguished Name (RDN), e.g. connectionId=123. This should be unique for all the object instances at the first level below a containing instance. If these

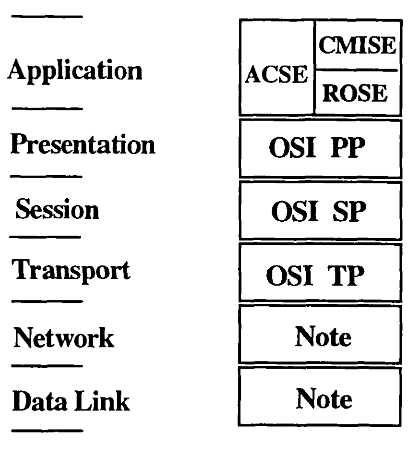
[1] The angular brackets indicate a set, since *allomorphs* is a set- or multi-valued attribute.

instances belong to the same class, then it is the value of the naming attribute that distinguishes them i.e. the "key".

The containment schema is defined by the name-binding GDMO templates which specify the allowable classes in a superior/subordinate relationship and identify the naming attribute. Name bindings and naming attributes are typically defined for classes in the first level of the inheritance hierarchy, immediately under top, so that they are "inherited" by specific derived classes. An example of a containment tree is shown in the right part of Figure 2-3, modelling connections contained by protocol entities, contained by layer subsystems, contained by a network element. A managed object name, also known as a Local Distinguished Name (LDN), consists of the sequence of all the relative names starting after the root of the tree down to the particular object, e.g. `{subsystemId=network, protocolEntityId=x25, connectionId=123}`.
OSI management names are assigned to objects at creation time and last for the lifetime of the object. An OSI managed object has exactly one name, i.e. the naming architecture does not allow for multiple names.

2.2.4 The Access Paradigm

In OSI-SM environments, managing functions, or objects, implement management policies by accessing managed objects. By access paradigm, we mean the access and communication aspects between managing and managed objects. Access aspects include both the remote execution of operations on managed objects and the dissemination of notifications emitted by them. OSI-SM follows a protocol-based approach, with a message-passing protocol modelling operations on managed objects across a management interface. The operations and parameters supported by the protocol are a superset of those available at the managed object boundary, with the additional features supporting managed object discovery and multiple object access. The protocol operations are addressed essentially to the agent administering the managed objects which acts as a naming, trading, notification and object access server as already discussed.



SE:	Service Element	PP: Presentation Protocol
ACSE:	Association Control SE	SP: Session Protocol
ROSE:	Remote Operations SE	TP: Transport Protocol
CMISE:	Common Mgmt Information SE	

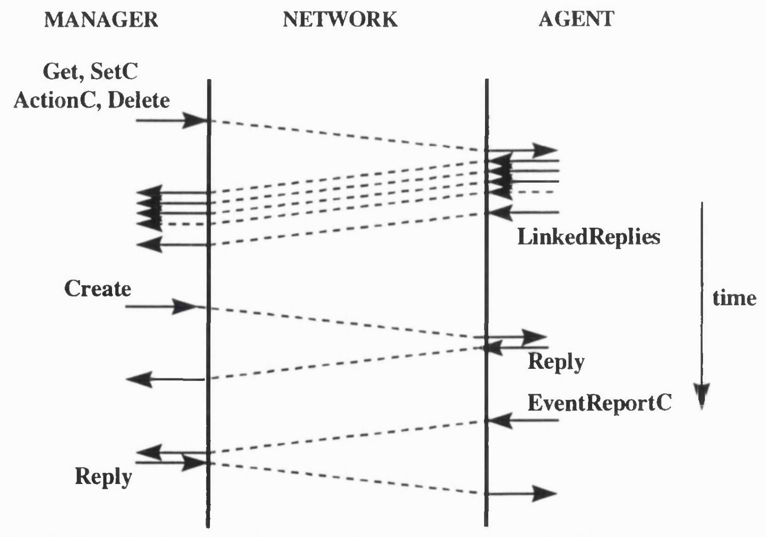
Note: There exist many OSI Network/DataLink protocol combinations

Figure 2-4 OSI-SM Protocol Stack

OSI-SM was designed with generality in mind and as such it uses a connection-oriented reliable transport. The relevant management service / protocol, the Common Management Information Service / Protocol (CMIS/P) [X710/11], operate over a full 7 layer OSI stack using the reliable OSI transport service; the latter can be provided over a variety of transport and network protocol combinations, including the Internet TCP/IP using the RFC1006 method [Rose87]. A more detailed view of the CMIP protocol stack is presented in Chapter 3 while a simplified view is depicted in Figure 2-4. End-to-end interoperability over networks with different combinations of data link and network layer protocols is supported either through network-level relaying or transport-level bridging as specified in [Q811]. The upper layer part is always the same and comprises the OSI session and presentation protocols, while the application layer part comprises the Association Control Service Element (ACSE) and the CMIS Element over the Remote Operation Service Element (ROSE) [Q812]. The benefit of a robust and reliable protocol stack is out-weighted partially by the fact that full 7 layer infrastructure is required even at devices such as routers, switches, etc., which typically run only lower layer protocols. In addition, application level associations need to be established and maintained prior to management operations and event notifications.

The OSI management model distinguishes between *systems* management, which uses the full OSI 7 layer stack, and *layer* management, which may use shorter stacks for efficiency or because higher layer communications are broken [Kler88]. Though layer management exists as a concept in the OSI framework, it has not been populated by layer management protocols. The only effort to date is CMIS/P Over Logical Link Control (CMOL) [Black92]. This is a lightweight CMIS/P-based link layer management protocol that attempted to dethrone SNMP from the private network market without success. The TMN has adopted OSI *Systems* Management as its base technology, so we are examining OSI-SM in this section.

Given the richness and object-oriented aspects of the GDMO object model, CMIS/P can be seen as a "remote method execution" protocol, providing a superset of the operations available at the object boundary within agents, with additional features to allow for object discovery, bulk data retrieval, operations on multiple objects and a remote "retrieval interrupt" facility. The CMIS operations are Get, Set, Action, Create, Delete, Event-report and Cancel-get. The Get, Set, Action and Delete operations may be performed on multiple objects by sending one CMIS request which expands within the agent through the scoping and filtering parameters. Since OSI managed objects are named according to containment relationships and organised in a management information tree, it is possible to send a CMIS request to a base object and select objects contained below that object through scoping. Objects of a particular level, until a particular level or the whole subtree may be selected. The selection may be further refined through a filter parameter that specifies a predicate based on assertions on attribute values, combined by boolean operators. Scoping and filtering are very powerful and provide an object-oriented database type of functionality in OSI agents. This results in simplifying the logic of manager applications and reducing substantially management traffic.



Note1: Get, Set, Action, Delete may also operate on one only object (single reply)
Note2: Set, Action, EventReport have also a non-confirmed mode of operation

Figure 2-5 CMIS Interactions

When applying an operation to multiple objects through scoping and filtering, atomicity may be requested through a synchronisation parameter. The result/error for each managed object is passed back in a separate packet, which results in a series of linked replies and an empty terminator packet. A manager application may interrupt a series of linked replies through the Cancel-get facility. Finally, the Set, Action, Delete and Event-report operations may also be performed in an unconfirmed fashion. While this is typical for event reports (the underlying transport will guarantee their delivery in most cases), it is not so common for intrusive operations as the manager will not know if they succeeded or failed. Nevertheless, such a facility is provided and might be used when the network is congested or when the manager is not interested in the results/errors of the operation. Figure 2-5 depicts the interactions between applications in manager and agent roles using CMIS, apart from Cancel-get.

The event reporting model in OSI management is very sophisticated, allowing for very fine control of emitted notifications. Special support objects known as Event Forwarding Discriminators (EFDs) [X734] can be created and manipulated in agent applications in order to control the level of event reporting. EFDs contain the identity of the manager(s) who wants to receive notifications prescribed through a filter attribute. The filter may contain assertions on the type of the event, the class and name of the managed object that emitted it, the time it was emitted and other notification-specific attributes, e.g. for an attributeValueChange notification, the attribute that changed and its new and old values. In addition, an emitted notification may be logged locally by being converted to a log record. The latter is contained in a log object created by a manager, which contains a filter attribute to control the level of logging. In summary, OSI management provides very powerful mechanisms to deal with asynchronous notifications and reduces substantially the need for polling-based management.

An example

We will consider now a concrete example in order to demonstrate the use of the OSI-SM access facilities. Assume we would like to find all the routes in the routing table of a network element that "point" to a particular next hop address. The manager application must know the logical name of the network element agent which it will map to the presentation address by using directory-based distribution facilities, which are explained in detail in section 2.4.1. The manager application will then connect to that address and request the relevant table entries through scoping and filtering in the following fashion:

```
Get ( objName={subsystemId=nw,protEntityId=clnp,tableId=route},
       scope=1stLevel, filter=(nextHopAddr=X),
        attrIdList={routeDest, routeMetric} )
```

The results will be returned in a series of back-to-back linked replies, as shown in Figure 2-5. The overall CMIS traffic will be kept fairly low: N linked replies for the matching entries together with the request and the final linked reply terminator packets i.e. N+2 in total. The overall latency will be slightly bigger than that of a single retrieval. It should be noted that traffic would be much more without filtering, since the manager would have to retrieve all the routing entries and perform the filtering locally. Finally, association establishment and release is also necessary to the element agent. This does not happen though on a per operation basis but associations may be "cached".

The manager application would also like to be informed of new route entries "pointing" to the next hop address X. This could be done by using the event reporting facilities provided by OSI management. The manager will have to create an EFD with filter:

```
(eventType=objectCreation AND objectClass=routeEntry AND nextHopAddr=X)
```

It will also need to set as destination its own name. After that, notifications will be discriminated within the agent and the ones matching the filter will be forwarded to the manager. Note that if there is no association to the manager, the element agent will establish it by going through the same procedure and mapping the logical manager name to an address through the directory. The previous observations about association caching and address mappings also hold in this case.

2.2.5 Generic Management Functionality

One key aspect of the OSI-SM framework is it follows a "large common denominator" approach to management standardisation, resulting in a large set of *common* object specifications that should be globally supported. A number of generic management functions are addressed in order to provide a well-defined framework for dealing with common tasks and to achieve reusability and genericity. These specifications emanate from the five functional areas (FCAPS) and are collectively known as the Systems Management Functions (SMFs) [SMF].

We can classify the OSI SMFs into four distinct categories:

i. those that provide *generic* definitions of management *attributes, notifications* and *actions* only; these are the Object Management [X730], State Management [X731], Relationship Management [X732] and Alarm Reporting [X733] SMFs;

ii. those that provide *system* definitions which complement the management access service by providing a controlled mechanism to deal with notifications; these are the Event Reporting [X734] and Log Control [X735] SMFs; and

iii. those that provide *miscellaneous* definitions that are used to support the management system itself; we can categorise here the security-related SMFs: Access Control [X741], Security Alarm Reporting [X736], and Security Audit Trail [X740]; and

iv. those that provide *generic* definitions of *managed object classes* that are used for common management tasks.

A full list of the current OSI SMFs is shown in Table 2-1.

ITU-T I ISO Number	**Systems Management Function**
X.730 I 10164-1	Object Management Function
X.731 I 10164-2	State Management Function
X.732 I 10164-3	Attributes for Representing Relationships
X.733 I 10164-4	Alarm Reporting Function
X.734 I 10164-5	Event Management Function
X.735 I 10164-6	Log Control Function
X.736 I 10164-7	Security Alarm Reporting Function
X.740 I 10164-8	Security Audit Trail Function
X.741 I 10164-9	Objects and Attributes for Access Control
X.739 I 10164-11	Metric Objects and Attributes
X.738 I 10164-13	Summarisation Function
X.742 I 10164-10	Accounting Meter Function
X.745 I 10164-12	Test Management Function
X.737 I 10164-14	Confidence and Diagnostic Testing
X.746 I 10164-15	Scheduling Function
X.744 I 10164-18	Software Management Function
X.743 I 10164-20	Time Management Function
X.748 I 10164-22	Response Time Monitoring Function
X.750 I 10164-16	Management Knowledge Management Function

Table 2-1 OSI Systems Management Functions

Starting from the third category first, the relevant functionality is important for security of management. These functions support controlled access to management information [X741], support the generation of security alarms when relevant breaches are detected [X736] and provide the framework and mechanism to conduct security audit trails [X740].

The functions of the second category are extremely important since they provide the means for *event-driven* as opposed to *polling-based* management. We term these *system* definitions since they make it possible to deal with notifications emitted from managed objects and control their forwarding to applications in manager roles through the Event-report CMIS primitive. As such, they complement the OSI System Management access service. You may recall that in the discussion of section 2.2.1 on the Management Functional Areas, event reporting and logging facilities were identified as an important requirement in all the five functional areas.

Functions of the first category are particularly important. Object Management [X730] provides three generic notifications related to configuration management that all OSI managed objects should support, namely object creation, object deletion and attribute value change. State Management [X731] provides a number of generic state attributes (administrative, operational, usage state, etc.) and a state change notification. It also prescribes state transition tables according to a well-defined state model. Relationship Management [X732] defines a relationship model based on pointer attributes i.e. attributes whose value is the distinguished name of another object. Relationships may also be represented by separate objects, as proposed by the more recent General Relationship Model (GRM) [X725]. Finally, Alarm Reporting [X732] provides a set of generic alarm notifications which cover the requirements for all possible types of alarms: quality of service, communications, equipment, environmental and processing error alarm.

Other MIB specifications should use the above definitions to model object, state, alarm and relationship aspects. Generic configuration, state or alarm managers may be written in a fashion that makes them independent from the semantics of a particular MIB. For example, a configuration monitor could be an application that connects to managed elements and requests all the object creation, deletion, attribute value and state change notifications in order to display changes to the human manager. Such an application can be written once and be reused: it only needs to be "told" the formal specification of the element MIBs in order to be able to display meaningful information for the objects emitting those notifications. OSI management platforms typically provide a set of generic applications that are based on those common specifications.

The relevant GDMO/ASN.1 specifications in the first and second category SMFs are compiled together in the Definition of Management Information [DMI] recommendation, which also specifies the properties of Count, Gauge, Threshold and Tide-Mark generic attributes.

Finally, functions of the fourth category constitute the majority of SMFs and provide a host of generic functionality. We describe the most important ones in more detail and the rest only briefly below.

Monitor Metric objects [X739] allow the observation of counter and gauge attributes of other MOs and their potential conversion to a derived gauge, which may be statistically smoothed. That gauge nay have associated threshold and tidemark attributes which support fully event-driven performance management capabilities, relegating "polling" totally within a managed element. Multiple input metrics provide the facility to combine different attributes in order to produce a comparison rate, e.g. for error vs. correct packets. Summarisation objects [X738] allow a manager to request a number of attributes from different objects of a remote system to be reported periodically, possibly after some statistical smoothing. These attributes can be specified using scoping and filtering while intermediate observations may be "buffered". This facility is important for gathering historical performance data for capacity planning and is typically used together with logging. The author has done research that builds on the metric monitoring and summarisation functions and provides a more flexible Intelligent Remote Monitoring framework [Pav96b].

Accounting Metering [X742] provides generic objects to support data collection for resource utilisation. Test Management [X745] defines generic test objects to provide both synchronous and asynchronous testing facilities. The target is to model generic aspects of testing so that they are separated from specific aspects. A number of specific Confidence and Diagnostic Testing objects [X737] have been specified using the previous framework, namely connectivity, data integrity, echo, resource, loopback and protocol integrity tests. Scheduling Management [X746] provides generic scheduler objects which could schedule activities of other MOs that support scheduling on a daily, weekly, monthly or a periodic basis, e.g. event forwarding discriminators, logs etc. Response Time Monitoring [X748] supports performance management by allowing the measurement of protocol processing time and network latency between network nodes. Software Management [X744] allows the delivery, installation, (de-)activation, removal and archiving of software in a distributed fashion.

Management Knowledge Management [X750] provides mechanisms for the dynamic acquisition of shared management knowledge, including *repertoire* knowledge (which managed object classes, conditional packages and name bindings a system supports), *instance* knowledge (the names of the available managed object instances) and *definition* knowledge (the formal MIB specification in GDMO). This knowledge is provided both through managed objects and through OSI Directory [X500] objects. These directory objects support also global naming and distribution facilities as it will be discussed in detail in section 2.4.1. The author has contributed research work towards the distribution and discovery aspects of [X750].

The OSI SMFs were depicted in Table 2-1. Note there exist more SMFs under development e.g. Domain and Policy Management [X749], Changeover Function [X752], Command Sequencer [X753] etc. In summary, SMFs increase the intelligence and capabilities of applications in agent roles. They provide useful generic facilities which most agent systems should support and enforce a common style of operation which may result in generic managing. The relevant objects are known as Support or Systems Managed Objects (SMOs) [X701] since they describe resources of the management system itself. In Chapter 3 we propose an object-oriented realisation model for the OSI SMFs that results in software reuse and makes available the relevant expressive power to management applications at a minimal cost.

2.3 The Telecommunications Management Network

This section provides an introduction to the Telecommunications Management Network (TMN) [M3010] which is the framework developed by the ITU-T for managing telecommunications networks and services. This is more than just an overview since it positions the TMN in the context of the Broadband Integrated Services Digital Network (B-ISDN) Reference Model [BISDN], presents the underlying requirements and explains the various architectural choices.

While the basic TMN recommendations have been available since 1990, there have been rather few tutorial or survey papers and relevant books in the literature. [Sahi88] provides an early view of the emerging architectural framework; [Bern93] discusses general issues related to the management of telecommunications networks; [Shrew95] provides an interesting overview ("TMN in a nutshell") but is biased towards the Network Management Forum OMNIPoint interpretation of TMN [OmniPnt], [Murr95]; finally [Glith95] and [Sidor95] provide easy-to-read tutorial overviews. In terms of books, [Aida94] is a collection of chapters by different authors addressing TMN issues, in which [Sahi94] and [John94] address the framework and standards bodies work. [Cohen94] is a chapter in [Slom94] that provides a fairly complete but rather dry overview of the TMN architectural framework.

The material in this section is based to a large extent on [Pav94a], [Pav96a] and [Pav97c].

2.3.1 The TMN in the Broadband ISDN Context

The purpose of a Telecommunications Management Network (TMN) [M3010] is to support operators in managing telecommunications networks and services. This means support for planning, provisioning, installing, maintaining and administering these. Management refers to set of capabilities related to the five functional areas described in section 2.2.1 and it is an integral part of the operation of networks and services.

Management becomes particularly important for the new generation of broadband telecommunications networks based on the Synchronous Digital Hierarchy (SDH) / Synchronous Optical Network (SONET) transmission and Asynchronous Transfer Mode (ATM) switching. These technologies offer advanced multiplexing capabilities that need to be exploited and harnessed through sophisticated management systems. Another important aspect that makes TMN a necessity is the advent of a new generation of telecommunications services that break away from basic telephony. Advanced services supporting multimedia, multi-party and mobility features need comprehensive service management facilities and Quality of Service (QoS) control, especially as they may be sold according to QoS agreements. The TMN provides a framework that tries to address these issues through a management model based on hierarchical decomposition and abstraction and through the deployment of object-oriented information specification and access principles that result in reusability and genericity.

The TMN is a data network carrying management traffic. It is logically separate from the telecommunications network being managed but interfaces to it at several distinct points in order to monitor and control its operation. A TMN may use, for its communication requirements, parts of the telecommunications network itself. The basic concept behind the TMN is to provide a framework in order to achieve the interoperation between management applications, which in TMN language are called Operations Systems (OSs), and the telecommunications equipment being managed. This is achieved through an architecture with standardised interfaces that support object-oriented message exchanges over well-defined protocol stacks. It should be possible for operators (or administrations in TMN parlance) to put together TMN systems that comprise equipment and operations systems from different suppliers, reducing costs and achieving rapid deployment and support for new services.

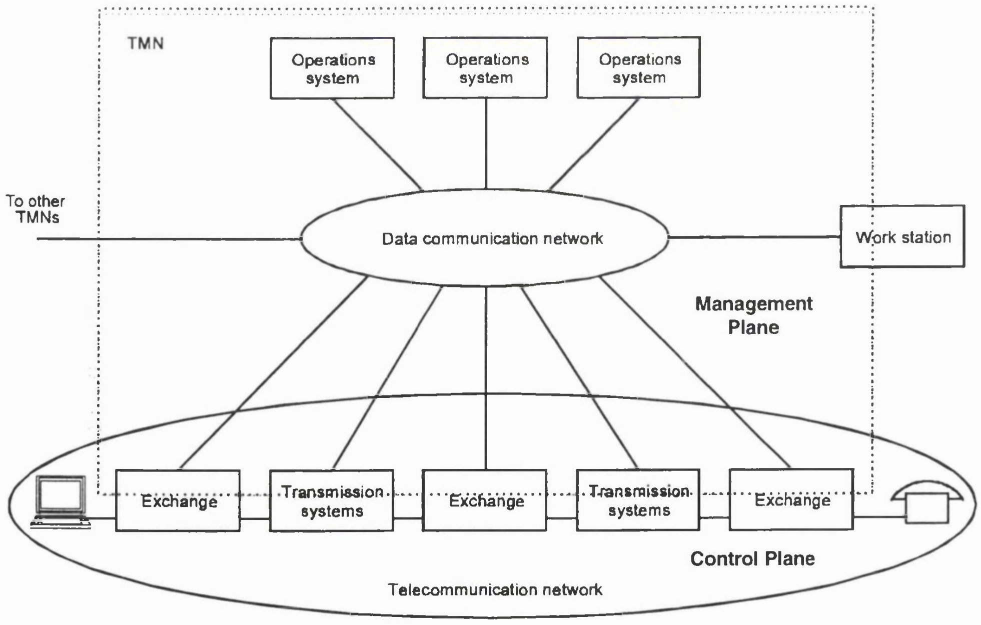


Figure 2-6 TMN Relationship to a Telecommunications Network (from [M3010])

Figure 2-6 shows the relationship between the TMN and the telecommunications network being managed. The user plane of the current generation of telecommunications networks supports raw transmission only, so a data network capability needs to be provided over them for communicating management traffic. This capability can be provided through the control plane over the Signalling System Signalling System No. 7 (SS7) protocols [Moda90]. If ATM becomes the core technology of the next generation as part of the Broadband ISDN [BISDN], the telecommunications network will be inherently a data network. TMN management applications (operations systems and workstations) operate typically on general purpose computer systems which are either directly attached to the telecommunications network or reside in local networks attached to it through interworking units. Although in Figure 2-6 the TMN boundary is shown as restricted to managing equipment in the telecommunications network, in reality it may also extend to manage equipment in the customer premises e.g. computer terminals for multimedia services.

The distinction between the TMN and the telecommunications network it manages, as depicted in Figure 2-6, maps exactly to the distinction between the *management* and *control* planes of the Broadband ISDN Reference Model [BISDN]. In the latter, activities in the operation of telecommunications networks are categorised in three *planes*:

- the *User Plane*, which is involved with the transfer of user information (audio and video streams, packet data);
- the *Control Plane*, which is responsible for the establishment, operation and release of on-demand calls and connections; and
- the *Management Plane*, which is involved with the planning, installation, configuration, provisioning and supervision of the network infrastructure in order to allow the user and control planes to function as efficiently and smoothly as possible.

According to this categorisation, the control plane supports end-user services and has also two distinct aspects. Support for *bearer* services, such as basic telephony through the Signalling System No. 7 (SS7) [Moda90] or, in the future, support for bearer ATM virtual channel connections through B-ISDN signalling [Q2931][Q2761]. And also support for *enhanced* services, such as Intelligent Network (IN) [Q1200] based telephony or multimedia services based on the emerging Telecommunications Network Information Architecture (TINA) [TINA]. The key difference between bearer and enhanced services is that all the intelligence for the former lives within the switching systems (the signalling protocols) while in the case of enhanced services the relevant intelligence operates in general computing systems attached to the network e.g. the service control point of IN or the service session control in TINA. As far as the TMN is concerned, both switching equipment and general purpose computing systems supporting advanced services are treated as managed elements.

From the above discussion it is should be clear that the TMN does not aim to support *on-demand end-user* services which are the subject of the control plane. The TMN supports management services which are primarily used by the operators and human managers of the telecommunications network. It also supports management services which are used by end-users, such as service subscription, accounting, service profile customisation and the provision of Virtual Private Network (VPN) [Reil96] management services.

The relationship between the TMN and the control plane bears a very good analogy to the relationship between a managed object and the associated real resource. The TMN provides management functions and facilities in an orthogonal fashion to the operation of the control plane but these management functions affect the way in which the control plane operates. The TMN influences operation of the latter by configuring operational parameters, for example routing table entries, according to management decisions. The TMN monitors the network, makes decisions based on network conditions and other information, such as management policy and knowledge of future events, and feeds back management actions to the control plane in order to influence its future behaviour. This relationship allows the network to operate as intelligently as possible without burdening the network elements with sophisticated features. In essence, management intelligence operates "outside" the network, in a similar fashion to enhanced service control frameworks such as IN and TINA. Finally, it should be added that since the TMN is not involved in on-demand end-user service control, it has less stringent requirements on real time response than control plane functions.

Note: The terms control and management defined above are sometimes used in a different fashion. By *control* it is meant functions embedded in network elements that are implemented in hardware/firmware, use signalling protocols and provide fast real-time response. By *management* it is meant functions operating in general purpose computing equipment, realised as software programs, which use general purpose data communication protocols and provide slower response. This is a physical categorisation as opposed to the functional one presented above. In the author's view, the former maps better to the B-ISDN reference model and it will be used consistently for the rest of the thesis.

2.3.2 TMN Architecture

In this section we examine the TMN architecture starting from the relevant requirements. We then present the functional architecture, explain the role of the various reference points and interfaces, discuss the logical layering aspects and finally examine the decomposition of an Operation System Function (OSF) into functional components.

Since we use the term *architecture* extensively in the rest of the chapter and the thesis as a whole, it is worth attempting to define it first. The definition is taken from the UK Alvey programme's Advanced Networked Systems Architecture (ANSA) project [ANSA89a]. There it is described as *"an engineering discipline of design with a common framework and a consistency of style"*. It embraces concepts and terms for explaining systems, models for reasoning about them, the specification of basic building blocks and a defined process and framework for putting systems together.

The TMN intends to support a wide variety of management activities which cover the planning, installation, operation, administration, maintenance and provisioning of telecommunications networks and services. The TMN should support the open exchange of information between management applications and managed elements and between management applications themselves. The latter should take place both within a TMN and across TMNs so that automated co-operation of different administrations is possible. Also customers will be offered access to management services through electronic interfaces.

Key requirements for TMN systems include the ability to:

- minimise the management reaction times to significant network events;
- minimise load caused by management traffic, especially when the telecommunications network is used to carry it;
- allow for geographic dispersion of management functions over a wide area e.g. a country-wide telecommunications network;
- be able to cope with very large scale sizes of manageable entities;
- provide automated isolation mechanisms for faults and invoke recovery procedures; and;
- improve service assistance and interaction with customers through electronic interfaces.

To provide the expected functionality and meet the relevant requirements, the TMN provides three architectural views [M3010]:

- The *Functional Architecture* refers to the distribution of TMN functionality into categories of "function blocks" interconnected via "reference points".
- The *Information Architecture* adopts OSI-SM object-oriented information specification and access principles. Managed resources are represented by managed objects at different levels of abstraction and are observed and modified by applications in managing roles in hierarchical or peer-to-peer manager-agent relationships. OSI-SM was explained in detail in section 2.2.
- The *Physical Architecture* maps combinations of functional blocks to physical blocks based on non functional requirements such as performance, ownership etc. In this view, some of the reference points become interoperable interfaces.

It is interesting to observe that the OSI-SM information specification and access framework has been adopted as the basis for the information architecture because it addresses some of the key objectives:

- It provides a highly configurable event model of fine granularity. This minimises reaction times to network events and keeps management traffic low with suppression of unwanted events at source.
- It provides intelligent information retrieval facilities based on scoping and filtering which keep management traffic low and provide better timeliness of management information.
- It allows for wide-area geographic distribution when combined with the OSI Directory Service [X500] which supports federation.
- It can cope with large scale of manageable entities in network elements as discussed in Chapter 3.

Despite the fact the information architecture currently uses OSI-SM principles, the TMN is not tied to the OSI-SM framework [X700]. Discussion is presently underway in the ITU-T Study Group IV to consider the use of Open Distributed Processing (ODP) [ODP] technologies, such as OMG CORBA [CORBA], as alternatives to OSI-SM. Chapter 4 of this thesis examines the feasibility of this assertion and proposes a solution.

2.3.2.1 Functional Architecture

The TMN management functions are realised by the following types or classes of function blocks:

- Operations Systems Functions (OSF),
- Workstation Functions (WSF),
- Mediation Functions (MF),
- Q-Adapter Functions (QAF), and
- Network Element Functions (NEF).

A *Network Element Function* models telecommunications and support equipment which communicates with the TMN for the purpose of being monitored and controlled. The NEF includes only those equipment aspects that are subject to management, i.e. the representation of relevant resources as managed objects and the necessary access mechanism. Control plane aspects are outside the scope of the TMN and, as such, are not modelled by the NEF. A NEF is typically an application operating in agent role according to the OSI-SM manager-agent model.

The *Q-Adapter Function* connects to the TMN those NEFs which do not support standardised TMN reference points. The responsibility of a QAF is to translate between a TMN and a non-TMN (e.g. proprietary) reference point. The main intra-TMN reference point is the *q* reference point, hence the name Q-Adapter. A QAF provides both protocol and information model translation and is typically a *proxy* agent-manager application according to the manager-agent model. Issues on Q Adaptation are discussed in more detail in section 2.4.2.

The *Mediation Function* acts on information passing between NEFs and OSFs in order to enhance the capabilities of "weak" NEFs. There are two types of q reference points, a fully capable q_3 and a "not quite q_3" one, called q_x. Mediation functions convert q_x to q_3 and may store, adapt, filter, threshold and condense information. They may be implemented in a hierarchical fashion in which q_x reference points are enhanced in various stages until q_3 is achieved. A MF may provide protocol conversion and information model translation and enhancement and is typically a proxy manager-agent application in a similar fashion to a QAF. Issues on mediation are discussed in more detail in section 2.4.2.

The *Workstation Function* provides the means for the human user to interpret TMN information, e.g. a network manager, and allow him/her to influence management decisions. The WSF interfaces to the OSF and MF and complements decisions based on human heuristics with logic that checks their validity and consistency. Note that it is not permitted for a human user to interface directly to a NEF or QAF, since relevant directives should always be checked by automated management logic. The WSF is typically an application in the manager role according to the manager-agent model. Issues on WSs are discussed in more detail in section 2.4.3.

The *Operations Systems Function* constitutes the heart of the TMN and processes information related to telecommunications management activities. OSFs access NEFs either directly or indirectly through MFs. They may also access *foreign* elements through QAFs. OSFs interact with each other in either peer-to-peer or hierarchical relationships. The latter are governed by the TMN logical layered architecture which prescribes a hierarchical structure in element, network, service and business management layers. The OSF is typically an application in both manager and agent roles according to the manager-agent model. The internal structure of the OSF is addressed in detail in section 2.3.2.5.

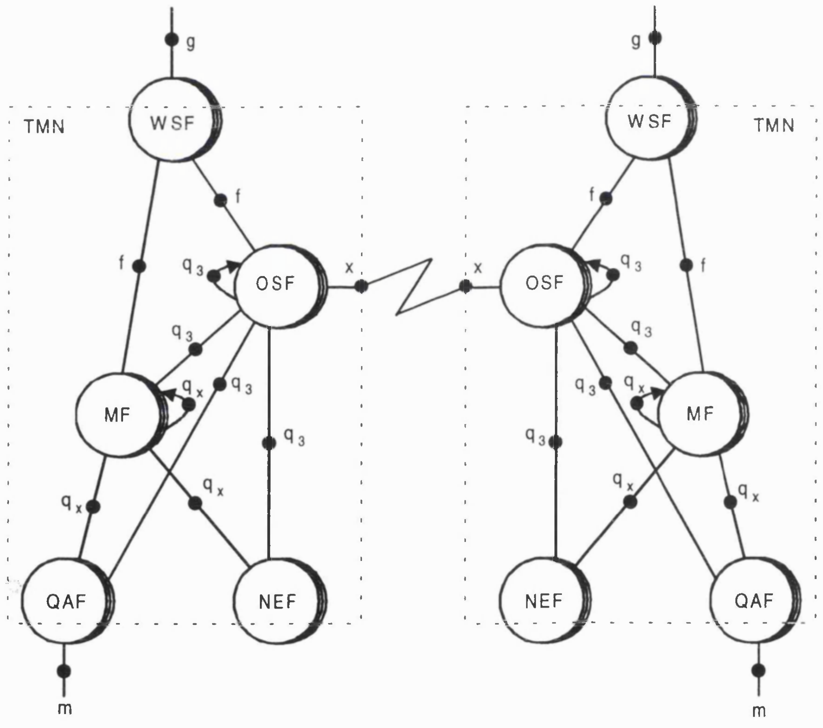


Figure 2-7 TMN Function Blocks and Reference Points (from [M3010])

In the discussion above we already introduced the q_3 and q_x reference points while discussing Q-adapter and mediation functions. The TMN defines a number of reference points which interconnect the function blocks presented above. The function blocks and reference points taken together define a *reference model* for the TMN. The logical functionality of any TMN can be described in a reference configuration by some arrangement of those function blocks and reference points. Figure 2-7 shows the relationship between function blocks and their interconnection by a number of reference points i.e. the TMN *Reference Functional Architecture.*

The TMN defines the following classes of reference points. Those inside or on the borders of the TMN (q_3, q_x, f and x) are subject to standardisation in order to support open management information exchange.

q_3: interconnects OSFs to NEFs, QAFs and MFs and OSFs to OSFs. It is realised through OSI application layer services: OSI-SM [X700], Directory [X500], File Transfer Access and Management (FTAM) [FTAM], Transaction Processing (TP) [DTP] and possibly Security [GULS].

q_x: interconnects MFs to "weak" QAFs and NEFs and MFs to each other in a hierarchical fashion. It is typically based on OSI-SM principles but "not quite".

x: interconnects OSFs to OSFs between TMNs. It is realised in a similar fashion to q_3 but with mandatory security mechanisms die to its inter-domain nature.

f: interconnects WSFs to OSFs and MFs. Very little work has yet been done towards its standardisation.

g: presents information to the human user in a graphical form. It captures the "look and feel" of the Graphical User Interface (GUI) together with its semantic specification e.g. a manual.

m: interconnects QAFs to *foreign* managed elements e.g. those supporting the Internet Simple Network Management Protocol (SNMP) [SNMP] or older telecommunications type management interfaces.

The q_3, q_x, x and f reference points are discussed in more detail later in this chapter.

2.3.2.2 Physical Architecture

Reference points define conceptual points of information exchange between function blocks. A functional block becomes a physical block and the relevant reference points become interfaces when that block is embodied in a separate piece of equipment, interoperating with other physical blocks over the TMN Data Communication Network (DCN). A *Physical Architecture* is the "instantiation" of a TMN system composed of physical blocks that conform to the reference functional architecture.

Interfaces are characterised by two facets, a conceptual facet and a physical facet, with the conceptual facet effectively defined by the reference point. The conceptual facet is characterised by the M-Part (Message Part) while the physical facet is characterised by the P-Part (Protocol Part) [Embry91]. In the case of OSI-SM which is currently adopted in the TMN, the M-Part defines the structure of the message sent to or received from a managed object i.e. a Common Management Information Service (CMIS) message [X710]. The P-Part defines the protocol stack used to transfer the message and this will involve the selection of the profile to support the Common Management Information Protocol (CMIP) [Q3]. Reference points are denoted by lower case letters (e.g. q_3) while interfaces are denoted by upper case letters (e.g. Q_3). The relationship between reference points and interfaces is shown in Figure 2-8.

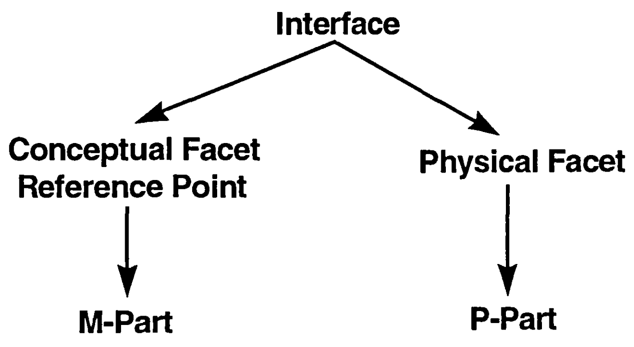


Figure 2-8 Relationship of Reference Points and Interfaces

It may be the case that more than one function block are combined together to form a physical block because of non-functional requirements. For example, two OSFs may constitute a single Operations System (OS), a MF and a QAF may constitute a Mediation Device / Q-Adapter (MD-QA) block, and so on. In this case their functional separation should still hold while their interactions should conform to a well-defined reference point. The actual realisation of that reference point becomes a proprietary issue since it remains internal within a network node.

2.3.2.3 Logical Layering

One of the key aspects of the TMN as a management framework is that it transforms the old *centralised* or *flat* management paradigm to a hierarchical distributed one (see also Figure 2-2 Models of Management Organisation). We will examine here the aspects and issues behind the TMN hierarchical decomposition.

In the hierarchical model projected by the TMN, management functionality is layered, offering increased abstraction and encapsulation in higher layers. The functionality of each layer builds on the functionality offered by the layer below. Each layer may contain additional sub-layers. The functionality of each layer (or sub-layer) is supported by a one or more Operation System Functions (OSFs). These may have peer-to-peer relationships when in the same layer or hierarchical ones when in different layers (or sub-layers). Each OSF presents an information model to superior OSFs which build on it and present different information models of higher abstraction to their superiors. As such, a layer in this architecture builds on the information model presented by the layer below and presents its own information model to the layer above.

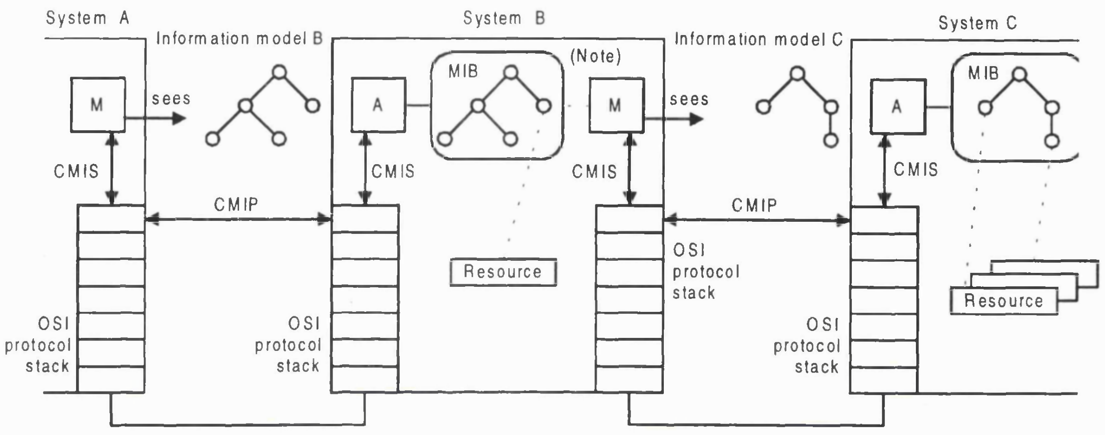


Figure 2-9 Cascaded Hierarchical Communication (from [M3010])

This hierarchical decomposition and the cascaded interactions in manager-agent roles with different information models at each level are depicted in Figure 2-9. The information models at each level represent manageable resources at different levels of abstraction. At the lowest level of that hierarchy, managed network elements contain resources of the finest granularity, representing aspects of the local system, e.g. access points, interfaces, call endpoints etc. At higher levels, objects represent resources of higher abstraction that can be mapped onto lower level resources, e.g. subnetworks, networks, services etc.

The TMN layers correspond essentially to different levels of abstraction in the information architecture. The mapping between information models at adjacent levels is accomplished through information conversion functions within OSFs. It should be noted that the use of OSI-SM as depicted in Figure 2-9 extends the OSI-SM model defined in [X701]. In the latter, managed objects are considered modelling resources in the various layers of the communications protocol stack. In the TMN this holds only for the lowest level of the management hierarchy i.e. the NEFs. Managed objects in higher TMN layers are more abstract views of the lowest layer objects.

TMN layering is not strict in the sense that an OSF may access directly OSFs in layers below the adjacent underlying layer. This may be the case, for example, between the service management and element management layers regarding historical QoS data, accounting records etc. The key aspect that characterises the TMN hierarchical organisation is that each layer is a fully incremental addition of the functionality provided by the layers below. As such, lower layers are essentially unaware of the presence of higher layers; they simply respond to management requests which are always initiated by the higher layers. Manager-agent interactions are only top-down or peer-to-peer in this layered hierarchy. Note that event notifications are always requested first (i.e. configured) by the superior system.

Another way to view the hierarchical nature of OSFs and the manager-agent relationships is depicted in Figure 2-10. This is known as the TMN Logical Layered Architecture (LLA). The LLA uses a recursive approach to decompose a particular management activity into a series of nested functional domains. Each functional domain becomes a management domain under the control of an Operation System Function (OSF). A domain may contain other OSF domains to allow further layering and/or may represent logical or physical resources as managed objects at the lowest level of the hierarchy within that domain. When the OSF domain is at the top of the layered architecture, there is no superior OSF.

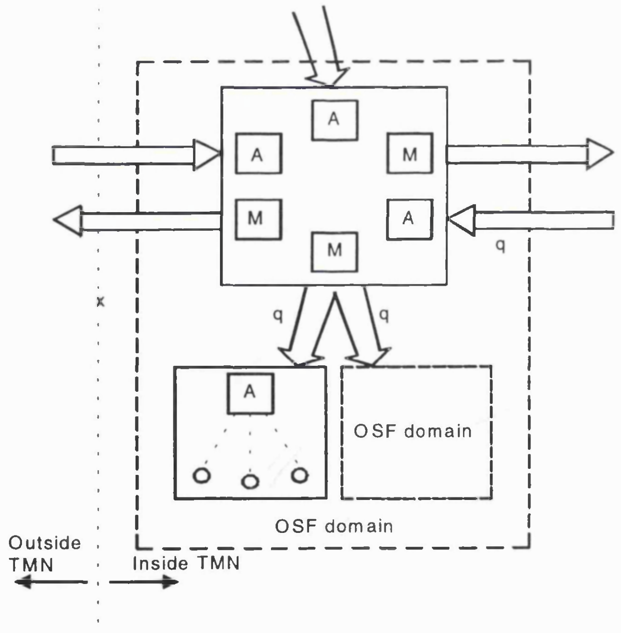


Figure 2-10 Logical Layered Architecture (from [M3010])

All the interactions within a domain take place at generic q reference points. Interactions between peer domains, i.e. crossing the OSF domain boundary can take place at a q or x reference point. Reference points may become interfaces depending on the physical realisation. The essence of the LLA is that at each point of interaction with an OSF domain, all the other subordinate OSF domains used to accomplish a management task can be thought as encapsulated (contained) by that domain. The user of a management service or sub-service offered by that OSF does not see its realisation.

The 1991 version of [M3010] indicated a potential decomposition of TMN layered functions in element, network, service and business management layers. These were influenced by British Telecommunications' Communications Network Architecture for Management (CNA-M) [Milh89]. This layered decomposition was only informative and not normative since the initial idea behind the TMN was that only Q_3 interfaces to network elements and X interfaces were to be standardised. In the 1996 version of [M3010] this layered decomposition became normative, which means that Q_3 interfaces within the TMN are also going to be standardised. The potential functions of the TMN layers of management activity are the following.

The *Element Management Layer (EML)* comprises functions related to either a single or a small number of network elements located in a small geographical area. These functions are concerned with the day-to-day management of elements in terms of configuration, faults and maintenance, performance monitoring, historical data etc. An abstract view of elements is presented to the network management layer so that the latter is shielded from unnecessary complexity and detail. These functions are predominantly technology dependent and do not take into account network-wide aspects as they have only a limited view of other parts of the network, if at all.

The *Network Management Layer (NML)* is concerned with keeping the network capabilities at some operational optimum. Human and automatic decision making processes may be used to optimise the network resources. This layer supports the management of the whole network which may span a large geographical area e.g. a whole country. The network is typically partitioned hierarchically into subnetworks, with complete visibility of the whole network provided at the top-most level. A technology independent view is presented to the service management layer. At this level, provision, cessation and modification of network capabilities are provided in order to support services. Statistical data are kept for the whole of the network concerning performance, usage, availability, faults etc.

The *Service Management Layer (SML)* is concerned with functions supporting services. Customer facing (provision, cessation, subscription, accounting, QoS, fault reporting) and interfacing with other administrations for the provision of end-to-end services are aspects of this layer. Additional tasks include the control of interaction between services, service order, complaint handling and invoicing. At this level, a technology independent view of the network is possible i.e. the network is treated as a "cloud" with a set of capabilities.

The *Business Management Layer (BML)* is concerned with the implementation of policies and strategies within the organisation which owns and operates the services and possibly the network itself. Business management functionality is typically proprietary and this subsequently means there is no x reference point at this level. Business decisions may be influenced by higher level controls such as legislation or macro-economic factors. They may address tariffing policies, quality maintenance strategies guidance on service operation, and so on. As an example, BT's "friends and family" package involves a business decision which affects the charging schemes in the service management layer and aims at attracting new subscribers (or keeping existing ones). It seems unlikely that the majority of business management functions will be automated in the near future.

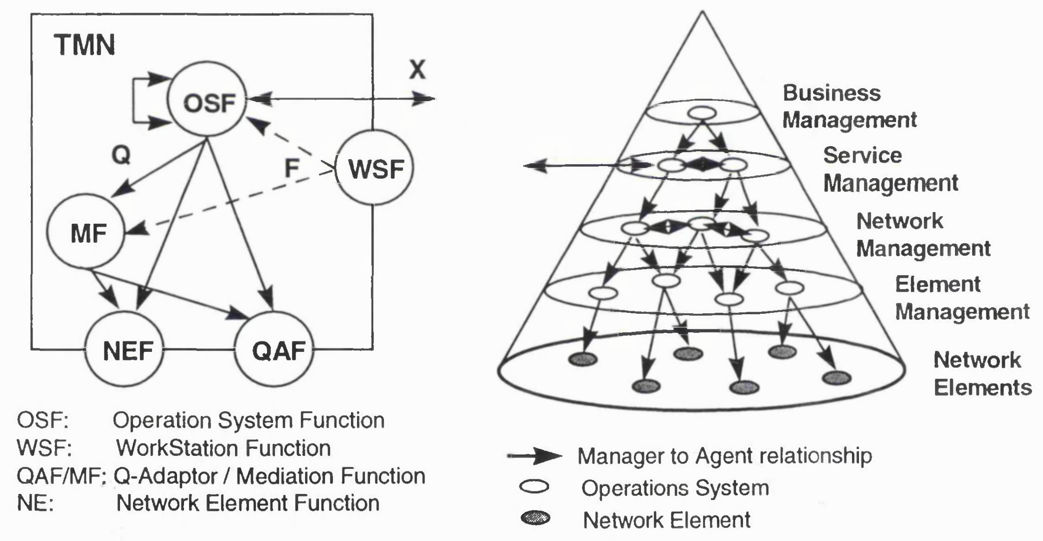

TMN
OSF
X
Q
F
WSF
MF
NEF
QAF
OSF: Operation System Function
WSF: WorkStation Function
QAF/MF: Q-Adaptor / Mediation Function
NE: Network Element Function
Business Management
Service Management
Network Management
Element Management
Network Elements
Manager to Agent relationship
Operations System
Network Element

Figure 2-11 TMN Functional and Example Layered Physical Architecture

Figure 2-11 shows in the left part a simplified view of the TMN functional architecture as presented in Figure 2-7 which maps to an example physical architecture in the right part; the latter demonstrates the TMN layers discussed above. Note that in the above example physical architecture, all the network elements are considered Q_3-capable i.e. there are no mediation devices or Q adapters; workstations have also been omitted for simplicity. The reason for the conical shape of the physical depiction is that higher sub-layers or layers comprise typically fewer OSs.

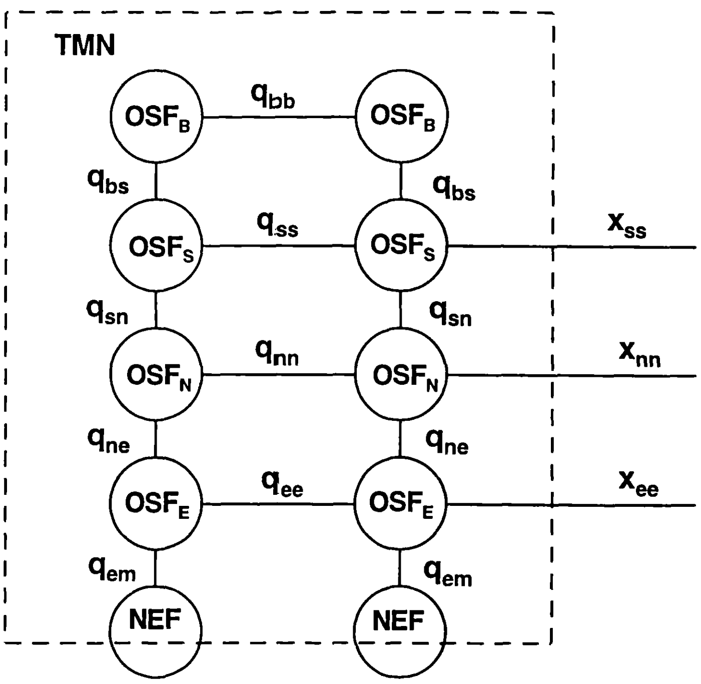


Figure 2-12 Classes of q and x Reference Points

The TMN functional architecture defines generic classes of q and x reference points for management information exchange. Given the hierarchical decomposition of the TMN into element, network, service and business management layers, specific classes of those reference points can be defined between layers and within a layer. Figure 2-12 depicts a hierarchical taxonomy of the relevant classes as presented in [GOM].

It should be noted it is unlikely that q_{ee} and x_{ee} will exist in reality. This is because element management OSFs typically follow strict hierarchical as opposed to peer-to-peer relationships. In addition, it may be the case that there will exist x reference points between different TMN layers and not only between the same layer. As an example, there may be network providers that offer pure transport only, with service providers buying those transport facilities to offer services. A service provider's TMN SM layer will need to interact with the network provider's TMN NM or even EM layers through x_{sn} and x_{se} reference points respectively. These types of cross-layer TMN interactions are not shown in Figure 2-12. Finally we mentioned previously that there may exist interactions between non-adjacent layers within a TMN e.g. between the service and element management layers. In this case, the q_{ne} reference point is used as "q_{se}".

An interesting aspect that deserves some discussion is that an intra-layer reference point, e.g. q_{nn}, is in principle the same with the inter-layer interface to the layer above, e.g. q_{sn}. This is because only one information model is typically standardised for the whole layer, addressing the "agent profile" to the layer above. The functional decomposition of a TMN layer into OSFs, and subsequently into physical OSs, is something outside the scope of TMN standardisation. The reason for this approach is that there are infinite ways to distribute the relevant functionality in OSFs. By leaving this unspecified there are no constraints on how to go about it and there is more scope for suppliers of TMN applications to diversify and compete with each other. On the other hand this implies that a particular operator is tied to buying the whole TMN layer from the same supplier in terms of the constituent OSs. This is because interoperability is actually dictated by the inter-layer and inter-TMN reference point specifications. It should be added though that international fora, such as the NMF, may extend the ITU-T recommendations and address the internal decomposition of a TMN layer in terms of well-defined OSFs.

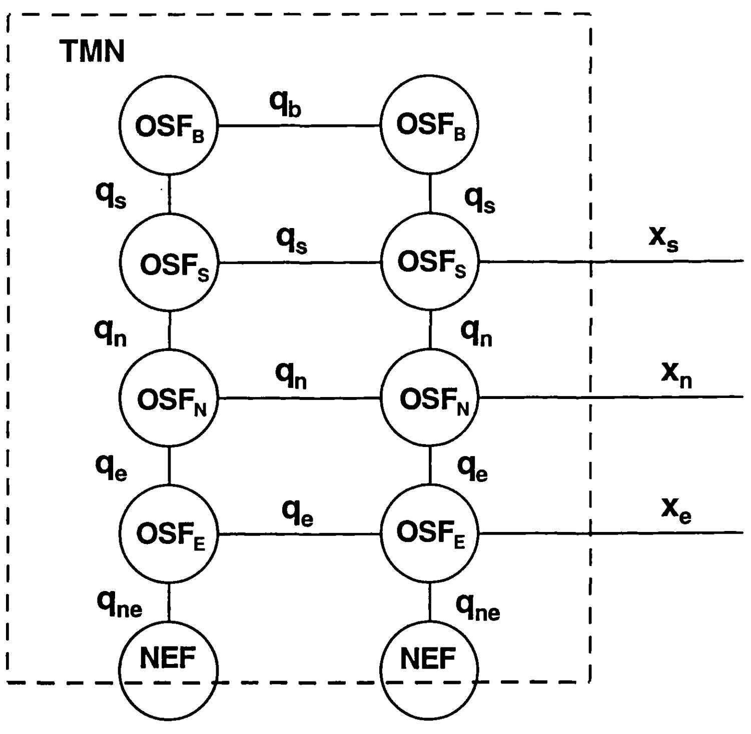


Figure 2-13 Updated Classes of q and x Reference Points

Given the previous discussion, it is obvious that it is the agent part that defines the functionality of a reference point. This means that the notation q_{ma} and x_{ma} in Figure 2-12 is equivalent to q_a and x_a. Taking into account that intra-layer and inter-layer reference points are the same, a simplified taxonomy of reference points is presented in Figure 2-13.

The reference points identified in Figure 2-13 are further specialised for a particular network technology and management service. At the time of writing (end of 1997), the following reference points have been addressed:

- $q_{ne\text{-}sdh}$: management for Synchronous Digital Hierarchy (SDH) elements - the ITU-T G.774 recommendation [G774].
- $q_{ne\text{-}atm}$: management for Asynchronous Transfer Mode (ATM) elements - the ITU-T I.751 recommendation [I751].
- $q_{ne\text{-}ss7}$: management for Signalling System No. 7 (SS7) elements - the ITU-T Q.751 recommendation [Q751].
- $q_{ne\text{-}gsm}$: management for Global System for Mobile (GSM) elements - the ETSI GSM12 specifications.
- q_n: technology-independent network layer management - the ETSI Generic Object Model (GOM) [GOM]and the equivalent emerging ITU-T network information model.
- $x_{s\text{-}coop}$ and $x_{s\text{-}user}$: inter-domain service management for path provisioning - the EURESCOM *Xcoop* (provider-to-provider) and *Xuser* (user-to-provider) specifications.

More reference points are being currently addressed by various groups.

2.3.2.4 Interface Specification Methodology

TMN interface specifications are typically produced by ITU-T, ETSI and NMF groups. They contain the formal description of managed object classes in GDMO together with usage scenarios that follow possibly the NMF "Ensemble" style [NMF-025]. The latter proposes scenarios that demonstrate how the supported management functions are accomplished through CMIS message exchanges with objects of the defined classes.

Interface specifications are produced in a top-down rather than a bottom-up approach. [M3020] describes a methodology to be applied when designing TMN interfaces. According to this, TMN designers start from the *management services* they would like to achieve, they decompose these to *management service components*, possibly recursively in more than one step. The finest granularity service components are further decomposed into *management functions,* which are then mapped onto managed object classes through an object-modelling phase. A consolidation phase is then necessary to make sure that the original management services are supported by the object classes the designer arrived at. The management information schema is finally produced in terms of name bindings for those classes. Additional notions such as the management context, management roles and management goals are also taken into account in the design process.

The management services and functions information model designers have come across are documented in [M3200] and [M3400], but these are informative rather than normative. Since TMN interfaces are object-oriented, new interface specifications should take into account the Generic Network Information Model [M3100] which specifies technology-independent generic classes to be specialised through inheritance. They should also take into account the functionality already offered by the OSI systems management functions [SMF].

[Sylor89] presents a set of guidelines for structuring manageable entities. RACE projects such as ICM [ICM] and PREPARE [PREPARE] have extended and refined the [M3020] methodology with the results fed back to the ITU-T. The author has contributed research work towards the ICM methodology described in [Grif96a].

2.3.2.5 Functional Components

[M3010] defines a number of *functional components* as constituents of TMN function blocks. These are not subject to standardisation but help to understand the internal functionality of the various function blocks. Reference points only are standardised and these are supported by Message Communication Function (MCF) components, which "understand" the structure of the relevant messages.

Since the Operation System Function is the most important and complex function block of the TMN, its structure in terms of functional components is shown in Figure 2-14. The Management Application Function in agent role, OSF-MAF(A), provides support for the information model exported across the q and x reference points. It receives messages that request operations on managed objects and returns results. It also emits asynchronous event reports according to defined preconditions in the relevant GDMO model specification. In a similar fashion, the Workstation Support Function (WSSF) understands the semantics of f messages and responds to relevant requests. The Security Function (SF) makes sure that only authorised entities communicate with the OSF (authentication) and controls the level of access to management information (access control).

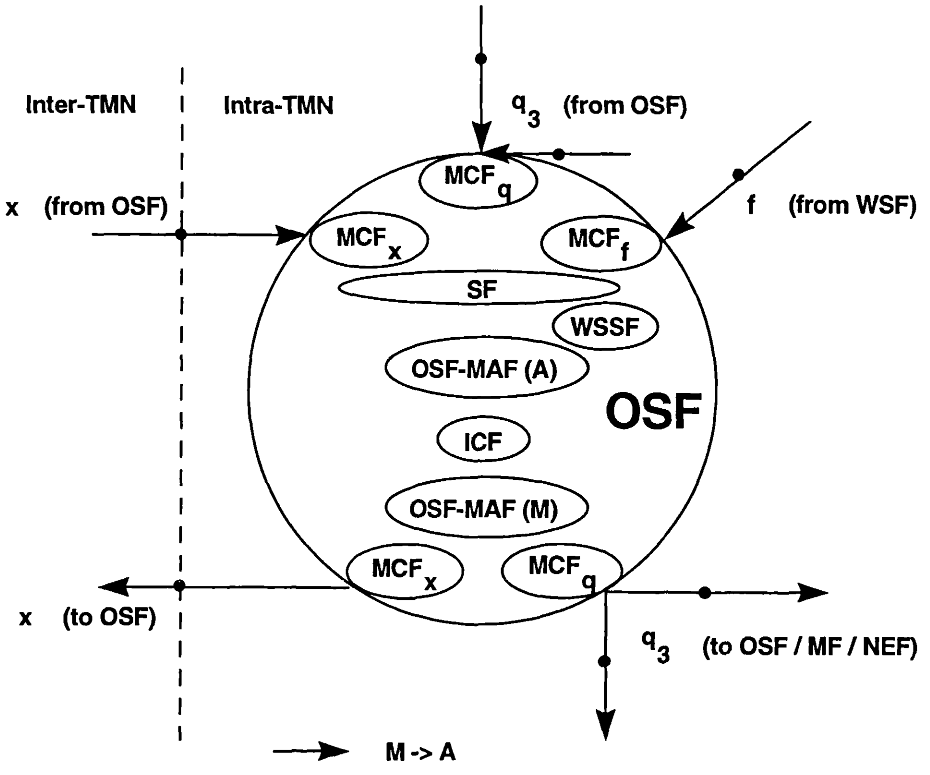


Figure 2-14 OSF Decomposition into Functional Components

The Management Application Function in manager role, OSF-MAF-M, provides the OSF's management intelligence by accessing subordinate information models in OSFs, QAFs and NEFs or peer ones in adjacent OSFs. It performs operations on managed objects and receives asynchronous event reports. Collection of statistics, alarm correlation and decision functions are typically located there. The Information Conversion Function (ICF) provides information translation functions that link the OSF-MAF-A and OSF-MAF-M. Operations on OSF-MAF-A managed objects may need to be mapped on operations to subordinate or peer information models. For example, a "path establishment" action on a network managed object in a network configuration management OSF will need to be mapped to operations on element manager OSFs to request relevant cross-connections. This mapping can be very complex since a relevant route needs to be identified that both satisfies the requested QoS characteristics (e.g. end-to-end latency, bandwidth, etc.) and can be accommodated through the existing resources. In the opposite direction, a link problem (e.g. fibre break) will cause alarms from many network elements since a large number of paths will typically be affected. Early correlation will take place in element manager OSFs but a network fault management OSF will have to perform the final correlation and generate a consolidated alarm announcing that the link is at fault. These are functions provided by the OSF-MAF-M and the ICF. They can be very complex, making use intelligent techniques such as rule based systems, learning processes based on neural networks, etc. This duality of operation in both manager / agent roles and the increased abstraction of management information in higher layers constitute in fact the essence of the TMN.

Table 2-2 presents the relationship of all the TMN function blocks to functional components. The MF and QAF are very similar to the OSF while the NEF acts in plain agent role. The WSF is the only different one, containing a User Interface Support Function (UISF) which requests and receives messages across the f reference point in order to drive the graphical display.

Function Block	**Functional Components**	**Associated Message Communication Functions**
WSF	UISF, SF	MCF_f
OSF	OSF-MAF(A/M), ICF, WSSF, SF	MCF_{q3}, MCF_x, MCF_f
MF	OSF-MAF(A/M), ICF, WSSF, SF	MCF_{q3}, MCF_{qx}, MCF_f
QAF	OSF-MAF(A/M), ICF, SF	MCF_{q3} or MCF_{qx}, MCF_m
NEF	OSF-MAF(A), SF	MCF_{q3} or MCF_{qx}

Table 2-2 Relationship of Function Blocks and Functional Components

2.4 Modifications and Extensions to the TMN Model and Architecture

2.4.1 Distribution and Discovery Aspects

The initial version of the TMN architecture as described in the 1991 version of [M3010] did not support distribution and discovery services. The defined reference points (q, x, f) supported only management information exchange, assuming information about the location and precise capabilities of other applications was acquired through ad-hoc mechanisms. Today's first generation of commercial TMN systems are still based on the 1991 TMN architectural principles and use static information in local databases to describe the location of applications. This approach makes it difficult to ensure global consistency and does not support distribution in terms of location transparency. Distribution transparencies are discussed in more detail in Chapter 4. [ODP] and in particular [X903] define location transparency as "hiding the topological details when identifying and binding to distributed objects" and [ANSA89a] defines it as "hiding the location of a distributed program component, enabling components to be located anywhere in a distributed system".

Given the fact that the TMN is based on OSI-SM technology, it is natural to try to use OSI naming services to support location transparency. In an OSI environment, the Directory [X500] provides a distributed naming service but, until 1993, it was not clear how to use the OSI Directory in order to name and locate TMN applications and associated managed objects. Research in the RACE ICM [Stath93] and PREPARE [Tschi93] projects and elsewhere [Sylor93] resulted in efforts to provide open mechanisms for distribution and discovery services. The author was substantially involved in the research and design behind the ICM approach which is described in detail in this section. The ICM and PREPARE approaches were harmonised and constituted the main input to the [X750] Management Knowledge Management Function. Before we examine the relevant issues, we will describe briefly the OSI Directory.

The OSI Directory [X500] provides a federated hierarchical object-oriented database. The latter lives in many Directory Service Agents (DSAs) that administer parts of a global Directory Information Tree (DIT). Parts of the global DIT belong to different Autonomous Administrative Areas (AAAs) and start at Autonomous Administrative Points (AAPs). DSAs are accessed by applications in Directory User Agent (DUA) roles via the Directory Access Service / Protocol (DAS/P) [X511] while DSAs communicate with each other via the Directory System Protocol (DSP). Accessing the local DSA is enough to search for information anywhere in the global DIT.

Figure 2-15 depicts the operational model of the directory and the global DIT. Directory Entries or Objects (DOs) are named using distinguished names that express containment relationships, in the same fashion as OSI managed objects. In fact, the directory naming architecture preceded that of OSI management and was essentially reused in the latter. The key difference between the Directory and OSI-SM information models is that directory objects are plain information objects containing only attributes i.e. they do not have behaviour, do not accept actions and do not emit notifications. A DIB is a fairly static database, i.e. directory objects do not change often, while a MIB is a dynamic management information store. Another key difference is the federated nature of the directory and the support of a unified DIB across all the DSAs.

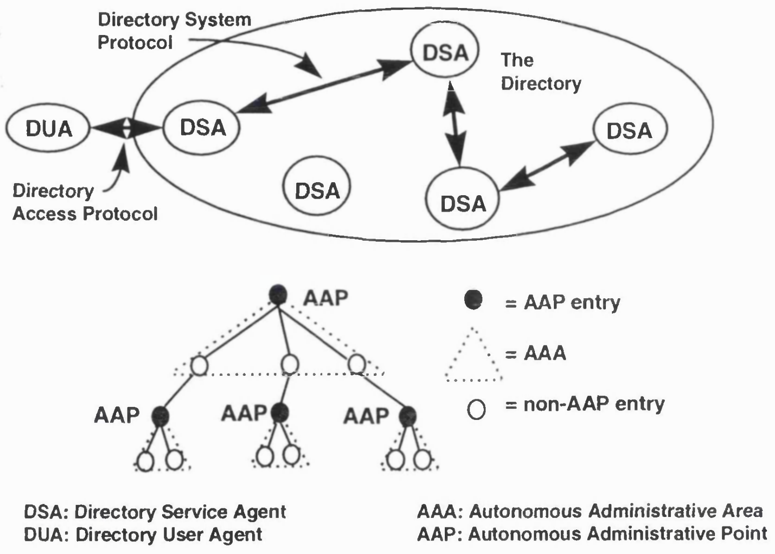

Directory System Protocol
The Directory
DUA
DSA
DSA
DSA
DSA
DSA
Directory Access Protocol
AAP
AAP
AAP
AAP
● = AAP entry
= AAA
○ = non-AAP entry
DSA: Directory Service Agent
DUA: Directory User Agent
AAA: Autonomous Administrative Area
AAP: Autonomous Administrative Point

Figure 2-15 X.500 Directory Organisational and Administrative Model

The Directory Access Protocol (DAP) bears many similarities to CMIS. There exist *bind* and *unbind* operations to support connect and disconnect operations respectively. The *read* operation is equivalent to a CMIS M-Get without scope and filter. The *list* operation returns the first level subordinates of a particular object, so it is equivalent to a M-Get with scope=1stLevel and no attributes requested. The *search* operation returns information from a number of objects according to scope and filter parameters, so it is generally equivalent to a M-Get. It should be noted that the equivalent "scope" parameter supports only the 1stLevel and wholeSubtree options i.e. there are no NthLevel and baseToNthLevel options as in CMIS. The *compare* operation compares a supplied attribute value with a directory attribute for which the user may not have access, so it is equivalent to M-Get with filter and no attributes requested. All the above operations may be cancelled when outstanding with the *abandon* operation, which is equivalent to the M-CancelGet. The *addEntry* operation is equivalent to M-Create and the *removeEntry* to the

M-Delete. Finally, the *modifyEntry* operation may be used to change the attributes of an object, so it is equivalent to the M-Set.

The first thoughts behind the issues of using the directory in an OSI-SM/TMN environment for naming, discovery and location transparency services took place during the first year of the RACE ICM and PREPARE projects in 1992. The partners involved in this work were UCL and ICS in ICM and GMD Fokus in PREPARE. UCL was actually a common partners in both projects, so it tried to achieve a harmonisation of the approaches. The common aspect is the representation of the TMN applications through directory objects and the unification of the directory and management name spaces, but there are also important differences. The PREPARE approach is described in [Tschi93] and [Bjer94]. Its key feature is the unification of CMIS/P and DAP as access methods through the *Management Information Service (MIS)*, with the directory used to store general management information of static nature e.g. TMN customer records and contracts. On the other hand, the ICM approach uses the directory *only* as a repository of information about the location and capabilities of TMN applications. The ICM approach is described in [Stath93], [Stath95], [Pav95a] and [Pav96a]. The common aspects of the two approaches became input to the [X750] recommendation and are described below.

A key aspect of the directory is its hierarchical naming architecture which is similar to that of OSI management. As such, it has been proposed to unify the two spaces in one global name space, considering management as an extension of the directory name space. The concept is depicted in Figure 2-16 which presents the extended manager-agent model through the use of the directory. Managed objects in an application in agent role, such as a TMN OS, may be addressed either through local names, e.g. {logId=1, logRecordId=1234}, or through global names starting from the root of the directory tree, e.g. {c=GB, o=UCL, ou=CS, cn=NM-OS, systemId=NM-OS, logId=1, logRecordId=1234}.

OSI-SM applications in manager, agent or hybrid roles are called Systems Management Application Processes (SMAPs) and are referred to through logical names e.g. NM-OS. Their management interfaces are known as Systems Management Application Entities (SMAEs) and each is associated to a, location dependent, OSI presentation address. The key issue for achieving location transparency is to decouple the SMAP name from the SMAE address and use the directory as a naming service that resolves a relevant name to the corresponding address(es). This can be done by modelling a SMAP through a directory object, the latter containing other directory objects modelling its SMAEs. SMAE directory objects contain addressing information as well as information regarding other aspects of that interface, termed Shared Management Knowledge

(SMK) [X750]. Since the same hierarchical naming architecture is used for both the OSI directory and management, the two name spaces can be unified. This can be achieved by considering a "logical" or "virtual" link between the topmost MIT object of an agent and the corresponding SMAP directory object.

The universal name space is shown in Figure 2-16 through the extended manager-agent model. The manager application may address objects through their global names, starting from the root of the directory tree, e.g. {c=GB, o=UCL, ou=CS, cn=NM-OS, systemId=NM-OS, logId=1, logRecordId=5}. The underlying infrastructure will identify the directory portion of the name i.e. {c=GB, o=UCL, ou=CS, cn=ATM-NM-OS}, will locate the relevant DO and will retrieve attributes of the contained SMAE DO, including the OSI presentation address of the relevant interface. It will then connect to that interface and access the required managed object through its local name i.e. {logId=1, logRecordId=5}. Note that applications in manager roles are also addressed through directory names for the forwarding of event reports, since the destination address in EFDs contains the directory name of the relevant manager.

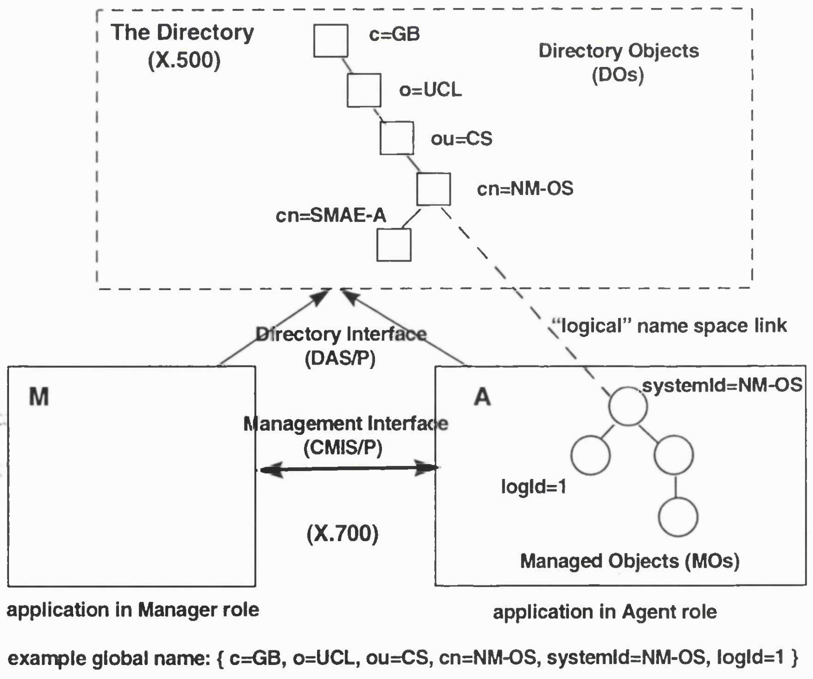


Figure 2-16 The Universal Directory and Management Name Space

[Sylor93] proposes a similar approach that uses a single directory object instead of the SMAP / SMAE separation. This directory object "mirrors" the class of the root MIT MO and its naming attribute, e.g. its class would be *system* in the above example instead of *applicationProcess* and the naming attribute would be *systemId* instead of *commonName* (or *cn* abbreviated). This object contains information related to the management interface, including the presentation address. Both the directory and the root MIT MO are called *junction* objects since they model the same entity and unify the two name spaces. An advantage of this approach is that the relative name of the directory application is not repeated in the global name e.g. {c=GB, o=UCL, ou=CS, systemId=NM-OS, logId=1}. On the other hand, a number of new directory classes need to be defined for every managed object class that can be in the root of a MIT e.g. system, managedElement etc. All of these would contain exactly the same information apart from the naming attribute and this is not a good object-oriented design practice. Another disadvantage is that an application is modelled only by one directory object and, as such, it may only have one management interface. This means that a hybrid manager-agent application with two interfaces needs to be modelled through two distinct directory objects.

The technique of unifying two name spaces that use the same naming principles is termed *grafting* in [Zatti94]. The latter contains also a nice tutorial overview of the OSI Directory and discusses in general issues of name management.

Let's examine now the mechanics of using the directory for naming in more detail. A management application needs to know beforehand the domain in which it operates, e.g. {c=GB, o=UCL, ou=CS}, its logical name, e.g. NM-OS, and the address of the local DSA. This information does not need to be hardwired in its logic but can be obtained from some local configuration repository. When it starts up, it forms its SMAP name and tries to access the relevant directory object through a *read* DAP operation. If it fails, it is probably the first time it is started up in that domain and it creates the SMAP and SMAE objects through *addEntry* operations. If it succeeds, it simply creates the SMAE objects for its interfaces. Finally, when it terminates, it deletes the relevant SMAE objects. If it terminates abnormally, e.g. it crashes, the SMAE object will contain inconsistent information until it is either deleted by a SMK "consistency checker" application or until the application is restarted. The SMAE relative name is a matter of local policy while parts of the presentation address, e.g. the selectors and ports, could be produced randomly to avoid clashes with other OSI applications running on that node.

When a management application needs to contact another one in order to perform operations on managed objects or forward event reports, it needs to know the SMAP name of the target application. It then has to perform a *search* operation, retrieving attributes of the SMAE objects in the first level below, including the *presentationAddress* and *mitMoList* attributes. If there exist two SMAE objects, one for an agent and one for a manager interface, it needs to identify which is the one it is looking for. This can be done by examining the *mitMoList* attribute, which contains the list of managed object classes the interface supports and the names of (static) object instances for every class [X750]. A non-empty mitMoList attribute signifies an agent interface. It should be noted that in the final version of [X750], the mitMoList attribute was left optional, despite the relevant complaints of the author and other ICM reviewers. This means that it may not be possible to determine the type of a management interface from the relevant directory information. In this case, trial and error is the only way to find this out. In the initial ICM approach [Stath93], there was an additional mandatory *managementRole* attribute that should take one of the values agent, manager and agent-manager. Unfortunately, this was not included in [X750].

In this universal naming scheme, managed objects may be addressed through global names which also implicitly denote the name of the application that contains the object. Global names guarantee location transparency since they remain the same even if the application moves because of availability, load balancing or other non-functional reasons. In this case, only the presentation address attribute of the relevant SMAE directory object needs to change. It should be noted that an application typically moves *within* an administrative domain e.g. within {c=GB, o=UCL, ou=CS}. When administrative boundaries are crossed, the name of the application and of the contained managed objects will have to change. This is acceptable since in this case a number of other things will need to change as well given the fact that the management system is restructured.

This naming architecture is built around the OSI-SM "managed object cluster" model, with a MIT handled by an application in agent role being the unit of distribution. A limitation of this architecture is that a managed object cannot move between agent applications and still retain its name. This is acceptable in the current TMN paradigm where applications contain managed objects that do not migrate. Future TMN systems may comprise *active* managed objects that act as mobile agents [Vass95][Vass97]. In this case, a migrating object will need to notify its clients when moving since its name is going to be different in the new hosting application.

We will now examine some aspects of this naming scheme in a TMN context. At the lowest level of the TMN hierarchy, network elements will have SMAP names denoting the particular element e.g. cn=switch-X. These applications will typically run always at the same network address but element managers will address them through their logical name, resolving it to an address through the directory. Q adapters adapting for a foreign network element should appear to element managers as if they were the element itself, e.g. cn=switch-Z, hiding the existence of the adaptation function. Mediation functions mediating for a number of elements should present separate views of the different elements, hiding the fact that the elements are dealt with together. In this case, there will be separate SMAP objects for each mediated element but the presentation address of the relevant SMAE will be the same i.e. the address of the node where the mediation device runs. The mediation device itself may appear as a separate SMAP through which it could be configured e.g. cn=MD-A. Finally, operations systems may take logical names such as cn=EM-OS-X or cn=NM-OS-Conf.

When a TMN OS is developed, it *sees* an information model in subordinate or peer systems and *supports* an information model for peer or superior systems as explained in 2.3.2.3 Logical Layering. During its development, no assumption should be made that instances of the relevant classes exist in a particular subordinate or peer system; this knowledge can be acquired dynamically, at run time. When it starts up, it should be "told" the SMAP names of the subordinate or peer systems to access, each of which may support a portion of the relevant information model. The relevant SMAE directory objects will contain the supported classes and possibly instance names, so the operation system will access the directory, retrieve this information and will be able to tailor its operation accordingly. As already mentioned in section 2.3.2.3, the TMN does not prescribe how to group physically the functionality of a TMN layer in terms of co-operating operations systems. The directory SMAE objects may be used to assist the dynamic discovery of the relevant information through the SMAE mitMoList information.

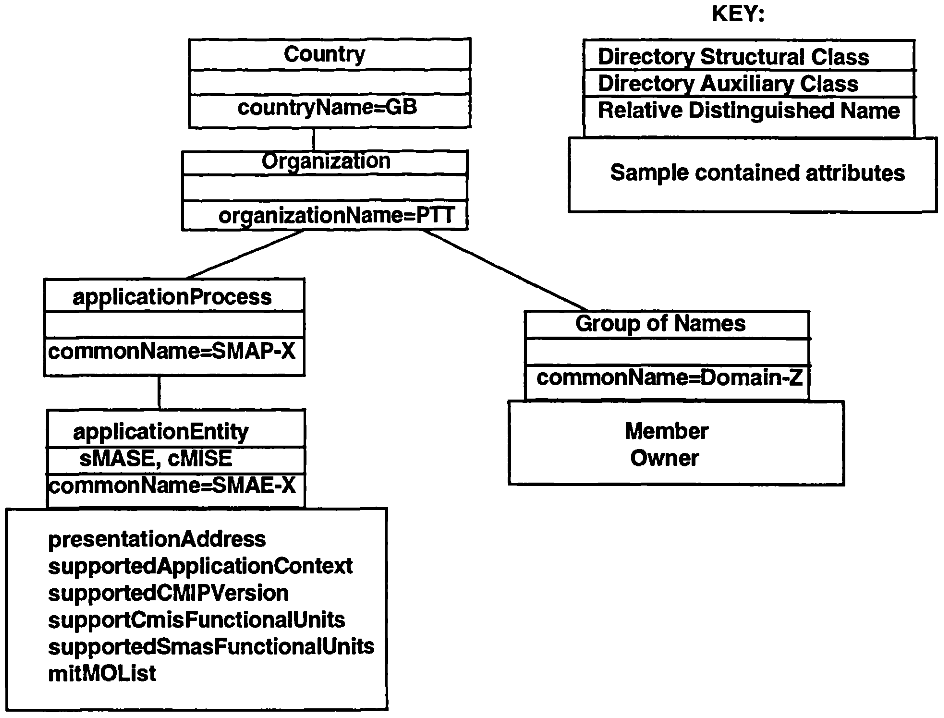


Figure 2-17 The Directory Schema for Distribution and Discovery (from [X750])

The described approach to distribution and discovery services for OSI-SM based environments was standardised in [X750] during 1995. The SMAP is modelled through the *applicationProcess* directory structural class and contains no significant information. The SMAE is modelled through the *applicationEntity* structural class and the *cMISE* and *sMASE* auxiliary classes. A relevant DO instance contains the presentation address and mitMoList attributes that have already been described. It also contains, among other, attributes describing the supported application context, the supported CMIP version, the supported CMIS functional units (scoping, filtering, cancel-get), and the supported SMFs (SMAS - Systems Management Application Service functional units). The directory schema for distribution and discovery is shown in Figure 2-17.

Note that application processes may be grouped together. This grouping is modelled through a *groupOfNames* directory object that contains the SMAP names of the group members. An application in manager role may be told initially the name of a groupOfNames object and access this one first in order to get the names of the SMAPs it should interact with. [Stath95] proposes a much more elaborate scheme for finding out dynamically which management systems contain the right managed object instances a manager application needs to work with. While such approaches are useful, it is very difficult to devise a truly generic scheme and standardise it. The approach taken in [X750] is that of a least common denominator. The scheme that was finally standardised may have minor deficiencies but goes a long way towards supporting distribution and discovery services in TMN environments.

An implementation of the initial ICM approach described in [Stath93] was undertaken in 1993, using the QUIPU directory system [QUIPU] [Kill91]. This was made available as part of the OSIMIS-3.3 release in early 1994. The system was re-implemented in 1995 to reflect closely the emerging [X750] standard and became part of the OSIMIS-4.1 release. The relevant realisation issues are discussed in Chapter 3. The TMN architecture [M3010] was also updated to suggest the use of the OSI Directory as the means for discovery and shared management knowledge services. The relevant implications on the TMN architecture are discussed in section 2.4.4.

While it has almost been three years since this approach was standardised by the ITU-T, the author is not aware of any commercial TMN applications that make use of the directory. Many TMN vendors talk of forthcoming support and it may happen given the recent revived interest in OSI Directory technology and the expressed commitment of companies such as Microsoft and Netscape. On the other hand, though the directory is another OSI application, it is *different* technology to OSI-SM. Vendors of OSI-SM platform infrastructure are not typically involved in the OSI directory market and do not have relevant know-how. This means that operators who deploy TMN systems need also to buy and operate directory technology separately. This increases both the cost and the complexity of a comprehensive TMN solution.

The reason the OSI Directory was chosen to support naming services was mainly because it provides federation and because it is considered complementary technology to OSI-SM. On the other hand, OSI-SM has all the relevant properties apart from federation and could be also used to support naming services. As an analogy, OMG CORBA uses naming services based on CORBA technology while it tries to solve the federation problem by other means. Had a similar approach been followed in the TMN, discovery facilities might have been commonplace in today's TMN systems.

Finally, it should be mentioned that despite the fact that the CMIP protocol [X711] does not mention federation across OSI-SM agents, a number of commercial TMN platforms offer such federation. This is simply achieved through special "junction" managed objects which are in effect logical links to the root MIT instance of a subordinate agent. The relevant agents treat those instances in a special fashion, forwarding requests to the subordinate agent by adjusting only the CMIS scope parameter so that the latter reflects the already searched levels [Pav97e]. A manager application may get the view of a global MIT which is in effect realised by different agents. In summary, OSI-SM *could* have been used to provide federated naming services, providing one homogeneous technology for the TMN and reducing both the complexity and cost of relevant solutions.

2.4.2 Issues on Adaptation and Mediation

TMN Adaptation and Mediation are similar concepts as they both perform information and protocol conversion and enhancement functions as discussed in 2.3.2.1. The key difference is that adaptation converts from reference points which are entirely non-compliant with the TMN i.e. various types of m, while mediation enhances various classes of "weak" q_x reference points to full q_3. Adaptation may yield either q_3 or q_x reference points; in the latter case, an additional level of mediation is needed to complement adaptation. Both adaptation and mediation are used to protect investment in network elements without TMN Q_3 interfaces.

2.4.2.1 Aspects of adaptation

The Q-Adaptor Function is used to connect to the TMN network elements that do not support the standard TMN interfaces. Q-Adapters provide information conversion functions from a proprietary information model and the associated access protocol to a TMN compliant information model and protocol. The proprietary information model may be object-oriented, simply object-based (e.g. SNMP) or there may be no distinct information model at all, which is the case with procedural management protocols. Examples of typical older telecom-type interfaces are TL-1 (Transaction Language 1) and TBOS (Telemetry Byte-Oriented Serial protocol). The Q-Adaptor Function should convert a proprietary m model to the TMN equivalent q_3 or a q_x one. Powerful querying aspects of the Q_3 access service such as scoping/filtering may need to be emulated through a series of operations across the existing M interface.

The nature of adaptation is depicted in the upper part of Figure 2-18. This is similar to the nature of class C mediation which is explained in the next section. Both adapter and class C mediation functions are hybrid units with respect to the manager-agent model: they access a lower level information model through m or q_x (MAF-M) and convert it to a q_x/q_3 or q_3 reference point respectively (MAF-A), through an Information Conversion Function (ICF).

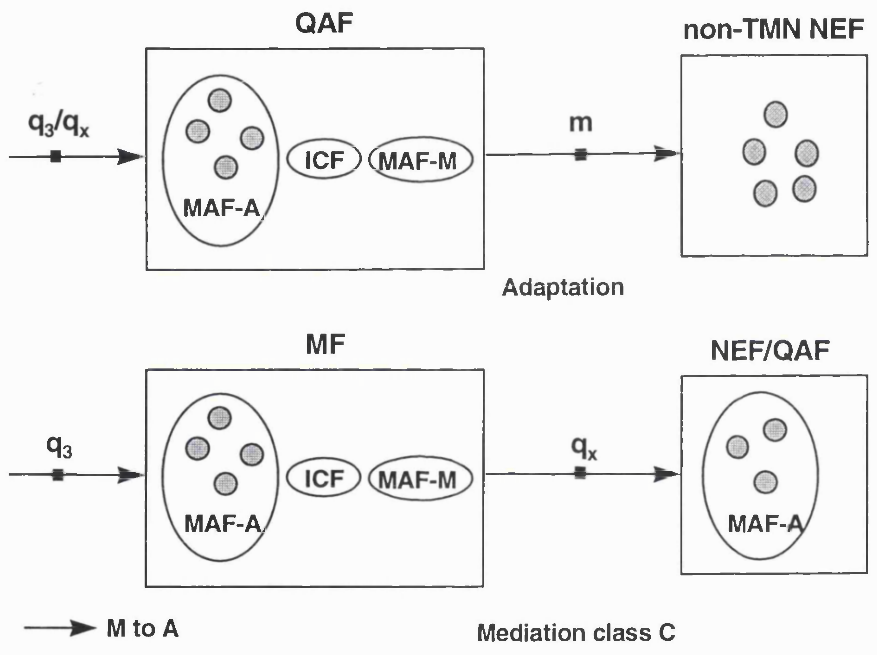


Figure 2-18 Adaptation and Mediation Class B

Ideally, automatic conversion between information models and associated access mechanisms is desirable in order to be used to automate the Q-Adaptation mechanism. This is only feasible for management frameworks of a similar nature. In principle, if Z is the target framework (i.e. OSI-SM in the case of TMN) and X a proprietary one, generic adaptation is possible *if and only if* X is a pure subset of Z in terms of expressive power, both in terms of the information model and the associated access service. In this case, X can be converted to the equivalent features of Z while some of the additional features may be provided in the adapter through mappings on simpler X features. Conversion in the opposite direction is problematic: some of the additional features of Z will remain unused while others can be mapped to equivalent X features only by using human heuristics.

A relationship between two management frameworks that has been extensively researched is that between OSI-SM and the Internet SNMP [SNMP]. A good general comparison can be found in [Geri94]. In this case, a pure subset relationship holds [Pav93c][IIMC]. The SNMP object model is a subset of the OSI-SM one, while CMIS/P offers access features that are a pure superset of the SNMP ones [Pav97a]. Additional features of CMIS/P such as scoping and filtering may be provided through multiple mappings on relevant SNMP features (get / get-next). The mapping in the opposite direction "wastes" OSI-SM facilities such as scoping, filtering and fine-grain event control. Additional problems are related to the conversion between the information models. Since SNMP does not support imperative actions, a GDMO action could be emulated through SNMP *set* and *get* primitives that pass the action parameters and retrieve the results. The mapping of the parameters and results to SNMP objects is something that can be done in many ways, requiring human intervention in the translation process and making difficult to automate adaptation units.

The author has been involved in early research investigating the relationship between the two frameworks and trying to identify generic rules for the unidirectional mapping between the two [Pav93c]. The latter involves the automatic conversion of a SNMP information model to the equivalent GDMO one and the automatic translation between CMIS/P and SNMP messages in the adapter unit that supports the resulting GDMO information model. This work culminated in relevant Network Management Forum specifications under the name ISO/ITU-T and Internet Management Coexistence (IIMC) [IIMC]. The author has also been involved in the design and implementation of a generic adapter that verified and validated the relevant concepts [McCar95]. The latter has been the first adapter of this type and has been made publicly available through the OSIMIS-3.3 (1994) and OSIMIS-4.0 (1995) [Pav95b] releases. It should be noted that only recently (1997) have similar products been announced in the marketplace.

Generic conversion rules between OSI-SM and any other non TMN compliant technology can result in fully automated adaptation which is economical to provide since it is not necessary to implement specific adapters for every interface of that technology. This makes possible the cost-effective integration of "foreign" elements and applications in a TMN environment. An important aspect is that the resulting "raw" information model from the automatic translation can be augmented and enhanced through the generic capabilities of the OSI Systems Management Functions [SMF], as for any other TMN interface.

The drawback of the generic adaptation approach is that the resulting information model, though semantically similar, is bound to be syntactically different to the equivalent TMN model due to the generic nature of the conversion. For example, translating the SNMP ATM element information model as in the RFC 1695 will result in a GDMO model that is syntactically very different to that of the equivalent ITU-T I.751 recommendation [I751]. The resulting reference point is q_x rather than a q_3 one as explained in the next section, so a further mediation step is necessary. Ideally, one would like to investigate generic mechanisms (e.g. a meta-language) that describe the relationship between a specific information model in two different frameworks and use them to automatically generate the adaptation unit; this is an interesting topic for further research.

While the TMN framework positions adaptation functions between NEs and Element Management OSs (2.3.2.1), in reality it may be the case that adaptation functions will take place elsewhere in the TMN hierarchy. For example, most elements with non-TMN management interfaces are typically supplied together with the associated element manager. Adapting for such elements means that new TMN-compliant EM-OSs need to be purchased, which is not cost-effective. An alternative solution is to convert the old element manager to a partly-compliant “EM-OS”, with a Q_3 interface to the network management layer, while retaining the M interface to the network element. This approach is proposed in [Glith96] and termed *Q-Addition*. Though not fully TMN-compliant according to [M3010], it is a pragmatic approach for retaining existing investment and migrating gradually towards a full TMN solution.

We close this section with a final remark on adaptation. Adaptation may be used to protect investment in non TMN-compliant elements and management systems if and only if the latter provide interfaces with ***similar*** expressive power to the TMN prescribed interfaces. In other words, if the relevant features are not present in the native “foreign” interface, they cannot be supported in the adapter. This implies that it may not be possible to fully integrate some of the older foreign elements into a TMN environment.

2.4.2.2 Aspects and taxonomy of types of mediation

According to the precise TMN definition, mediation functions may “adapt, store, filter, threshold and condense information”. A mediation function acts on information passing between an OSF and NEF or QAF and essentially enhances a q_x reference point to produce either a more capable q_x or a fully capable q_3 one. This procedure may take several iterations, with intermediate levels of q_x reference points and a recursive cascaded structure of the relevant mediation functions. This explains the MF to MF interaction in the TMN functional architecture of Figure 2-7.

The idea behind the q_x reference point is that it is simpler than the “fully capable” q_3 one and, as such, easier to provide in network elements with limited computing resources. In addition, q_x

would result in reduced investment in the relevant software/firmware development by equipment suppliers, enabling the early introduction of TMN-capable network elements. The difference between the q_x and m reference points is that the former is a TMN endorsed interface, using some form of TMN information model representation and access mechanisms (GDMO/CMIS). Various q_x reference points would be endorsed by the ITU-T, as proposed by equipment suppliers with relevant products.

There has never been a taxonomy of possible types of qx reference points and relevant interfaces, either by the ITU-T or in the literature, apart from [Pav96a]. The author proposes the following classes of Qx interfaces and, subsequently, classes of mediation:

A. The protocol stack is not standard in comparison to the general requirements for the Q3 interface [Q3] in general (Class A);

B. The CMIS [X710] and SMAS [SMF] capabilities are not fully supported as dictated by the relevant Q3 specification for that type of interface (Class B); and

C. The relevant information model is not fully standard, again as dictated by the relevant Q3 specification for that type of interface (Class C).

Class A points to data network protocol interworking. It should be noted that this is higher layer protocol interworking, any lower layer interworking due to different data network technologies that support the Q_3 interface [Q811] should be transparently addressed either by network layer relays or transport bridges. Examples of non-standard Q_x protocol stacks include CMIS/P Over LLC (CMOL) [Black92] or the early lightweight mapping of CMIS/P over TCP/IP (CMOT) [Besa89]. There exist also other non-standard mappings, used by various equipment suppliers.

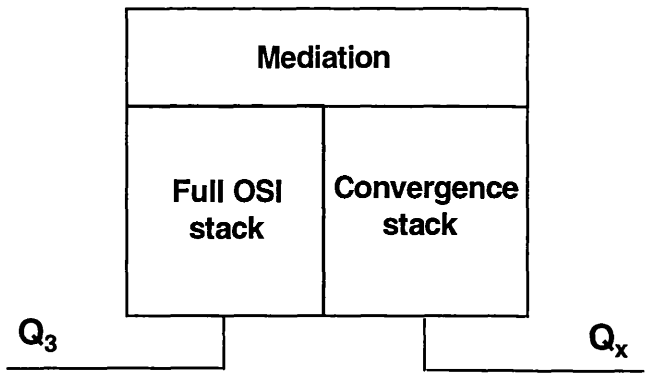


Figure 2-19 Mediation Class A

This type of mediation device provides either pure CMIS service conversion or combined service and protocol conversion. When the Q_x protocol stack provides equivalent functionality to the Q_3

one, e.g. connection-oriented reliable packet service, service conversion is only necessary. This means that CMIS and association control primitives are simply copied between the two stacks, in a trivial fashion. If the Q_x stack is of lesser functionality, e.g. connectionless unreliable as is the case with CMOL, then protocol conversion is necessary to provide reliability and appropriate mappings for association management. In both cases, the required mediation device is relatively simple. This type of mediation device is depicted in Figure 2-19. Note that a different mediation device is necessary for each different type of Q_x.

Mediation Class B is necessary to overcome the inability of Q_x to support required CMIS and SMAS capabilities: scoping and filtering [X710], storage (logging - [X734]), thresholding (metric monitoring - [X739]) and condensing (summarisation - [X738]) attributes of mediation. The functionality of this type of mediation device involves "mirroring" the relevant Q_x information model and enhancing it with the missing properties in order to become Q_3. This enhancement may take place through a series of cascaded devices, each possibly grouping a number of similar elements together. Since the functionality of the missing function is well-known, it is possible to produce generic mediation devices of this type. If we wanted to depict pictorially this class of mediation function, it would be similar to Class C (Figure 2-18) but the two information models would be exactly the same.

Class C means that the information model is different to the TMN prescribed one for the particular technology. This is typical of early TMN-capable elements, implementing early views of relevant developing standards with proprietary changes/additions to fill in gaps and to provide the desired functionality. The mediation device required in this case is depicted in the lower part of Figure 2-18. This class of mediation is similar to adaptation but the key difference is that the access mechanism is CMIS/P and the information model is described in GDMO/ASN.1. Converting this type of Q_x to Q_3 cannot be easily automated. Given the fact the two models are semantically equivalent, it should be possible to describe their relationship in a meta-language which could be parsed to produce the mediation logic, as discussed in the previous section. The problem is that this is a research problem that has not yet been solved. As a consequence, a specific mediation device needs to be developed for *every* network element with this type of Q_x interface. Given the fact there exist almost infinite variations of non-standard information models from different suppliers, it becomes both difficult and expensive to deal with this type of heterogeneity through mediation devices.

It should be finally mentioned that a Q_x interface may belong to more than one of the identified classes. Table 2-3 shows the various aspects of M, Q_x and Q_3 interfaces. M does not have any q reference point aspects (CMIS/GDMO) and, as such, it has none of the three interface aspects as well. Q_x has the q reference point aspects, though it does not have all the aspects of the Q_3 interface. As such, we can actually characterise 7 types of Q_x: one that has none of those three aspects, three that have one only and three that have two.

I/F characteristic	M	Q_x	Q_3
q reference point	x	m	m
a. standard CMIP stack	x	o	m
b. required CMIS and SMAS FUs	x	o	m
c. required GDMO model	x	o	m
	x: not present	o: optional	m: mandatory

Table 2-3 Characteristics of the M, Q_x and Q_3 Interfaces

In summary, Class A of mediation is relatively straightforward to provide since it only requires service and protocol conversion. A different type of such a mediation device is necessary for every type of Q_x protocol stack. Class B is more difficult to provide but, in principle, it could be dealt with in a generic fashion. Class C is problematic as there exist almost infinite variations of non fully standard Q_x models and it cannot be dealt with in a generic fashion. In any case, the existence of Q_x necessitates the use of mediation devices which require additional investment. Because of this, operators deploying TMN systems are typically locked into buying the EM-OS together with the Q_x-capable element, which is against the principle of a fully open, multi-vendor TMN system.

The above analysis was first published in [Pav96a] and was also passed as input to the ITU-T Study Group IV that addresses the TMN architecture. It was also proposed to avoid standardising the q_x reference point / Q_x interface and remove q_x from the TMN functional architecture together with the mediation function. The main reason for the existence of Q_x was the fact that Q_3 was considered expensive and heavyweight to provide circa 1990, when the first version of the TMN architectural framework was produced. The evolution of technology in the mean time, in terms of cheap processing power and memory, together with research for the provision of efficient TMN development environments such as OSIMIS, has invalidated to a large extent those reasons. Chapter 3 of this thesis shows that the overhead of a full Q_3 interface is much less than widely believed, both in terms of computing resources and in terms of required development time. As such, it is perfectly possible to provide direct Q_3 interface capabilities for telecommunications network elements. It still might not make sense to deploy a Q_3 interface in a LAN repeater or bridge, but telecommunications network elements are typically much more capable and sophisticated (exchanges, multiplexors, switches, intelligent peripherals etc.).

2.4.3 Workstations and the F Interface

Workstation functions (WSFs) provide the means to support human users of management services in realising management decisions or performing supervisory activities. They essentially provide the means to interpret TMN information to be presented to the human user and vice-versa. A separate reference point and corresponding interface (f and F respectively) realise the logical and physical communication between workstation functions and operations system or mediation functions.

According to the TMN functional architecture, workstation functions are not connected to either network element or q-adaptation functions. The reason for this restriction is that human users should not be allowed direct access to elements but only through the use of management functions provided by OSFs so that their decisions are implemented in a controlled fashion. [M3010] suggests that MFs may support decision functions and, as such, WSFs may connect to MFs through a f reference point (Figure 2-11). The analysis in the previous section suggests that there are no such aspects of mediation but only q_x to q_3 enhancement. As such, WSFs should *not* interface to MFs but only to OSFs.

While the f reference point and F interface have been an integral part of the TMN architecture from its early stages, there have never been serious efforts towards their standardisation within the ITU-T. The first tangible output has been recommendation [M3300] which explains further the workstation concept and presents a number of management capabilities required at the F interface in the various functional areas. The latest draft [M3010] reflects the most recent views of the ITU-T regarding the F interface. According to this, the relevant functionality is a superset of the Q_3 functionality in terms of the information model involved. Potential F information in addition to Q_3 managed objects could be display support (e.g. maps), information on GUI function and command initiation, help text etc. While this is the intention, it is a really challenging problem to propose an approach for F that is generic enough not to constraint implementations and also able to cope with the needs of different types of WSs.

Another reason behind the existence of the F interface was that WSs should be able to run on inexpensive equipment with limited resources e.g. laptop PCs etc. The protocol part of the F interface could be a lightweight mapping, supporting this capability. The analysis in the previous section for the Q_x interface holds also for F in this respect: inexpensive equipment does not necessarily imply limited resources anymore while, in addition, a full Q_3 protocol stack is not expensive in terms of required resources as it will be shown in Chapter 3.

In the absence of standards for the F interface, the following assumptions can be made:

- f, F are transaction-oriented in a similar fashion to q, Q; and
- interactions across f, F can be always mapped onto q, Q.

The key assumption is that all the necessary information to support workstation functionality can be derived from management information in Q_3 form. In other words, Q_3 models can be defined in such a way to be able to support all the necessary information for the human TMN user.

From an architectural point of view, this implicitly means that direct workstation access is allowed to any q-capable function block, including mediation, adaptation and network element functions. Given the previous discussion though, WSFs should have intrusive access only to OSFs: security services should be used to restrict direct intrusive access to MFs, QAFs and NEFs.

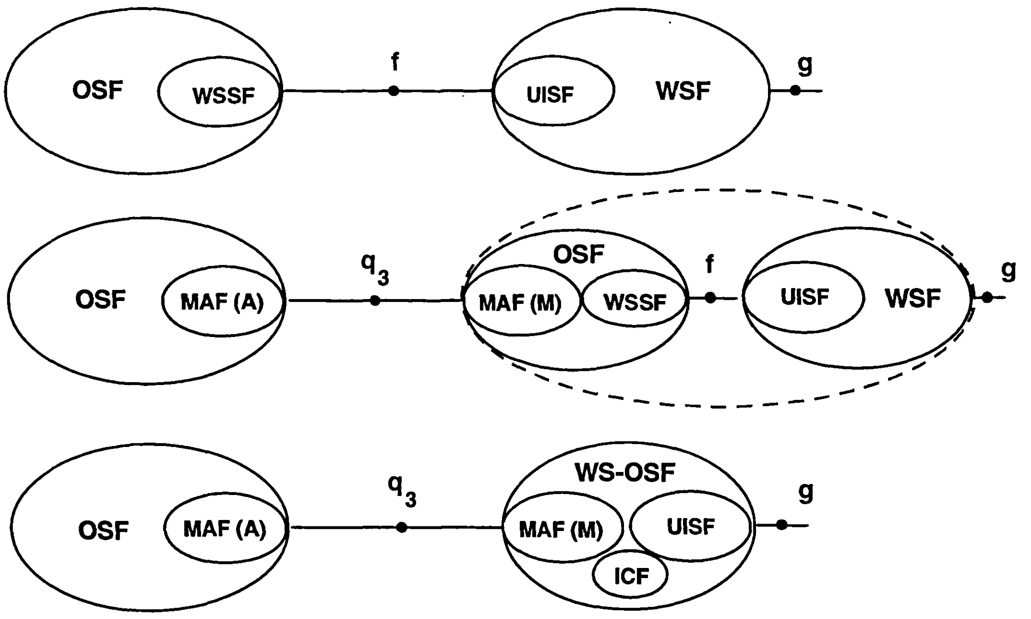


Figure 2-20 The WS-OSF Function Block

Given the fundamental assumption that f can be derived from q_3, we propose the configuration for workstation applications depicted in Figure 2-20. The upper part depicts the notion of a WS as in [M3010]. The mapping of f to q_3 can be supported by an OSF in managing role. The Workstation Support Function (WSSF) in the latter receives f messages and through a Management

Application Function in Manager role (MAF-M) converts these to q_3 messages (middle part of Figure 2-20). This mapping may not be one-to-one but should be always possible if the necessary information is available in q3 form. Taking this concept a step further, we introduce a new TMN function block called Workstation-Operation System Function (WS-OSF). The latter comprises a User Interface Support Function (UISF), an Information Conversion Function (ICF) and a MAF-M functional component and may interact with OSFs across q_3 (lower part of Figure 2-20). As a consequence, the TMN WSSF functional component is no longer necessary, being replaced by the ICF.

According to the approach described above, interoperability between WS-OSs in manager roles and OSs in agent roles is based on Q_3 interfaces. For this architectural modification we have also taken into account the fact that full Q_3 support is not expensive, as it will be shown in Chapter 3. The f reference point remains internal within a WS-OS and becomes a matter of design discipline rather than a TMN architectural prescription. In other words, it is up to the designer of a WS-OS application to separate GUI support functionality (UISF) from the application's management intelligence (ICF/MAF-M). Such a separation can be effected through a well-defined internal software interface (i.e. a f reference point) rather than a "on the wire" interoperable F interface. This separation provides for a cleaner design and makes possible to change the technology used for the GUI without affecting the rest of the WS-OS application. According to our experience, this separation is possible and desirable but it is not easy to make the f reference point independent of the application-specific functionality. This is one of the reasons that we believe interoperability should be based on the Q_3 interface and, as a consequence, we suggest that the f reference point *should not be standardised* [Pav96d].

Finally, it should be possible to run workstation displays on platforms without Q_3 stack support for reasons other than cost. For example, it should be possible to use a string-based CMIS encapsulated in Hypertext Transfer Protocol (HTTP) [HTTP] packets to drive WWW GUIs, etc. These are essentially class A Q_x protocols as explained in the previous section and a protocol converter is required to translate them to Q_3. Such converters are essentially "mediation functions" that operate in the opposite direction and can be thought as part of the broader WS-OS physical block. It should be finally mentioned that OSs might directly support such "Q_x" interfaces in addition to Q_3, which is the practice with a number of commercial products. In this case, the WS-OS is typically sold together with the vendor's OS.

In summary, we propose that ITU-T does not address the standardisation of the F interface so that full flexibility is possible in providing TMN workstations. The bottom line for interoperability should be the Q_3 interface, which has led us to define a more specific WS-OSF block that replaces the WSF block. As a consequence, the WSSF functional component is no longer necessary. The above analysis was first documented in [Pav96a] and [Pav96d] and has also been provided as input to the ITU-T TMN architecture group.

2.4.4 The Architecture Revisited

In the previous three sections we presented a number of modifications and extensions to the TMN architecture. These covered the following:

- the use of the OSI Directory for distribution and discovery services;
- the taxonomy of classes of mediation and the suggestion, after a relevant analysis, that q_x / Q_x should not be standardised; and
- the introduction of the WS-OSF block that replaces the WSF and the suggestion that f / F should not be standardised since they can be derived from q_3 / Q_3.

The impact of the last two on the TMN reference functional architecture is evident: no q_x and subsequently no MF; and no f and the replacement of WSF by the new WS-OSF block. The first one though suggests the introduction of a new technology in the TMN, namely the OSI Directory technology, and it raises the question of how the latter is integrated in the TMN from an architectural point of view.

Since the introduction of the directory to the TMN was initially proposed by the RACE projects ICM [ICM] and PREPARE [PREPARE], they both included relevant integration schemes. The PREPARE approach is documented in [Bjer94]. According to this, a new function block is introduced, the Directory System Function (DSF). All the other TMN function blocks connect to the DSF through a *d* reference point, which embodies discovery facilities through the directory access service. The drawback of this approach is that it does not address the DSF to DSF interaction either within a TMN or across TMNs. Is there another reference point to model the directory system service between DSFs? [Bjer94] stays moot on this point.

The ICM approach was developed by the author and is documented in [Pav96a]. It is similar to the PREPARE approach in terms of the new DSF function block but differs in the sense that access to the DSF is provided through the q_3 reference point within a TMN or through the x reference point across TMNs. These model, in this case, the directory access service. In addition, DSFs may communicate with each other either through q_3 or x, which in this case model the directory system service.

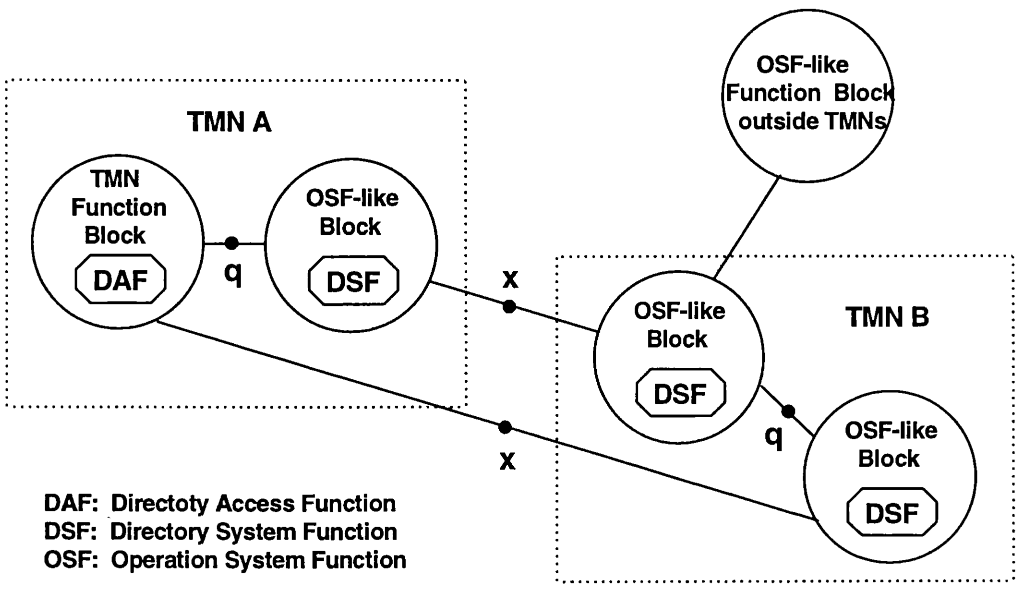


Figure 2-21 TMN and Directory Integration

Both above approaches were submitted to the ITU-T TMN architecture group which finally decided on a third variation that has minimal impact on the existing architecture, as depicted in Figure 2-21. This is closer to the author's approach in the sense that no new reference points are introduced but on the other hand it does not introduce a new function block. Directory system functionality is provided by OSF-like function blocks which model, in this case, passive directory databases. The relevant functionality is supported internally by the Directory System Function which becomes now a functional component. Other TMN blocks that need access to discovery services contain a Directory Access Function (DAF) component. Access to OSF-like directory blocks may take place within a TMN of across TMNs, in a similar fashion to the author's approach described above.

The approach adopted by the ITU-T will appear in the new version of [M3010] which is still in draft form (end 1997). The author feels this approach is elegant and general while it aligns better with the possibility of a TMN based on distributed object technologies instead of X.700/X.500. For example, a CORBA Naming Server [COSS] is a part of the distributed computing infrastructure, realised as a CORBA object while access to it is provided through the standard CORBA mechanisms and protocols. As such, naming and discovery services should not be treated in a special fashion through separate reference points and function blocks. Issues related to the architectural evolution of the TMN due to the future use of distributed object technologies are discussed in Chapter 4.

Given these modifications, we propose the revised TMN functional architecture presented in Figure 2-22. It is interesting to observe its simplicity compared to the original architecture presented in Figure 2-11. The key important changes are the removal of q_x and MF, the removal of f and the replacement of the WSF by the WS-OSF. A key point that needs to be emphasised is that now there exist special types of OSFs that act as directory servers. As a result of this, the q3 and x reference points are "overloaded" with directory access and directory system communication aspects. NEFs, QAFs, WS-OSFs and non-directory OSFs contact the "directory" OSFs to inform them of their location and capabilities or to discover the location and capabilities of other blocks they need to communicate with. Directory OSFs communicate with each other and with similar function outside TMNs to provide the unified global directory information tree.

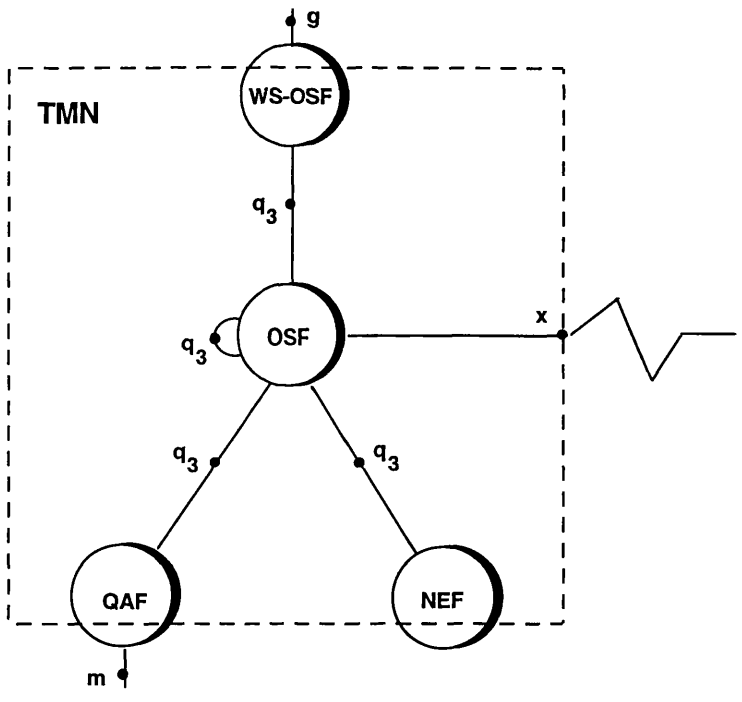


Figure 2-22 Revised TMN Functional Architecture

Table 2-4 shows the revised relationship between function blocks and functional components, updating Table 2-2. All the function blocks contain a DAF functional component while the OSF may contain also a DSF component when behaving as a directory server. Note also that there are no more WSSF, MCF_{qx} and MCF_f.

Function Block	**Functional Components**	**Associated Message Communication Functions**
WS-OSF	UISF, ICF, MAF(M), DAF, SF	MCF_{q3}
OSF	OSF-MAF(A/M), ICF, DAF, DSF, SF	MCF_{q3}, MCF_x
QAF	OSF-MAF(A/M), ICF, DAF, SF	MCF_{q3}, MCF_m
NEF	OSF-MAF(A), DAF, SF	MCF_{q3}

Table 2-4 Revised Relationship of Function Blocks and Functional Components

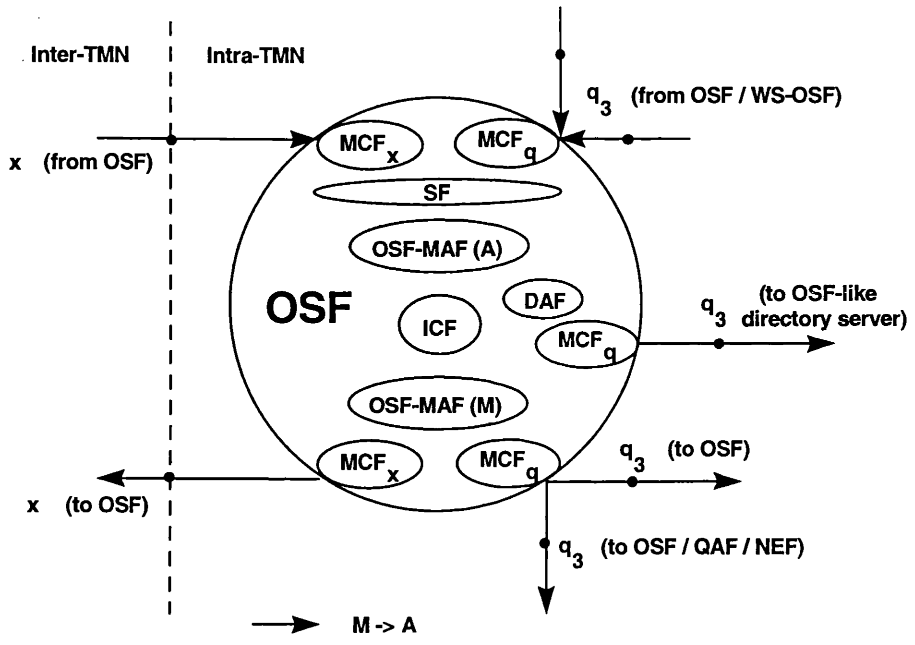


Figure 2-23 Revised OSF Decomposition into Functional Components

Figure 2-23 shows the revised decomposition of an OSF into functional components which was initially depicted in Figure 2-14. We are going to explain this in some detail. The WSSF component and the associated MCF_f that supported f message exchanges has been removed. A DAF component has been added that supports distribution and discovery functions by accessing OSF-like blocks that offer directory services via a MCF_q. The OSF-MAF-A implements the agent functionality of the OSF, supporting the managed objects required by the q_3 and x reference points. The same physical Management Information Tree (MIT) supports both q_3 and x interactions, though different views may be presented through access control functions [X741]. The latter are realised by the SF together with authentication, data integrity and possibly confidentiality services [GULS]. The SF essentially provides secure access to the MIT across the

q_3 and x reference points. The OSF-MAF-A needs to update the directory through the DAF about its location and capabilities when it is first started. Subsequently, it will have to resolve the names of other OSFs that appear as destinations in event forwarding discriminators to addresses through the DAF.

In the next chapter, we will show how the internal structure of an OS can be supported by a object-oriented development environment, achieving reusability, supporting a rapid development cycle and not sacrificing performance. We will then revisit the decomposition of Figure 2-23 and we will associate it to the relevant concrete engineering abstractions.

Coming back to the revisions we have proposed to the TMN architecture, it should be noted that the ITU-T has recently agreed to remove the q_x reference point and the mediation function from the forthcoming new version of [M3010] but they have not (yet) decided to remove the f reference point. On the other hand, the existence of very little work supporting the F interface and the proliferation of WSs based on various diverse technologies suggest a potential future removal. The initial TMN architectural framework was complicated and difficult to understand and explain. The proposed changes simplify it without sacrificing any important aspects for open interoperable telecommunications management.

2.4.5 Discussion

Having first introduced the TMN architectural framework and subsequently presented a number of modifications and extensions to it, we will now discuss some salient features and clarify further a number of issues. Let's first qualify exactly how a TMN interface is specified and how the overall capabilities of a TMN physical block should be expressed in terms of both exported and imported interfaces (the latter for blocks other than NE and QA).

A Q_3 or X interface has typically a manager-agent or client-server duality: an interface is exported by an application in agent role or it is imported by another application that acts in manager role. Interfaces are only specified from the agent or server side but the capabilities of a TMN application with dual nature such as an OS are expressed through both exported and imported interfaces; the latter are necessary for it to accomplish its management functions. As such, a TMN application with managing aspects other than NE/QA is qualified through the description of both exported and imported interfaces.

Since TMN interfaces are transaction-oriented, specifying the capabilities of a Q_3 or X interface has two aspects:

- specifying the entities that are made available across the interface in terms of their syntax and semantics, e.g. management and directory objects, files; and
- specifying the protocols used to convey these entities between systems.

The second aspect should specify the exact protocol stack profile. The upper layer stack profile is always the same [Q3][Q812] while it should be stated which of the lower layer profiles of [Q3][Q811] that interface supports. The first aspect should specify the management, directory and file information model for file transfer.

The management part of a Q_3 or X interface is specified in terms of a GDMO model which constitutes an "ensemble" [NMF-025], i.e. the relevant object instances collectively provide a management capability. The exact features implemented should be mentioned, i.e. optional aspects included etc. The latter information should be also made available through the directory as shared management knowledge.

The directory aspects of Q_3 and X involve the use of the directory service to either notify/update it with its location and capabilities or to learn about these with respect to other applications. Recall that all TMN applications have OSI Application Process names which map onto a unique directory name, e.g. {c=GB, o=UCL, ou=CS, cn=ATM-OS}. The directory information model for shared management knowledge activities has been specified in X.750, so it suffices to mention that an application supports it.

A key aspect in the TMN is that its functional entities are essentially composite computational components or building blocks for management services, presenting capabilities through a managed object cluster made available at the exported interface(s). Internally, within each such functional entity there may exist other computational entities which are not visible externally. These require additional object modelling by OSF designers. The properties of managed objects are specified in GDMO while the Object Modelling Technique (OMT) [Rumb91] may be used as an additional notational technique. Any internal computational entities may be also described using OMT. It should be added that the modelling and placement of computational logic can be arbitrary, in the sense that managed objects may also have computational properties and not be simple information objects. The computational aspects of all these objects, i.e. internal behaviour, state and response to stimuli can be specified using formal description tools such as SDL [Z100]. The mapping of these objects to concrete engineering depends on the relevant engineering infrastructure. Chapter 4 of this thesis proposes an object-oriented environment for the rapid and efficient realisation of TMN systems, in which managed and other internal computational objects are mapped onto C++ object instances.

One particularly important aspect of the object cluster model used in the TMN is the existence of generic functionality available at every management interface. This functionality is provided by the OSI Systems Management Functions (SMFs) which specify generic support managed objects that may be instantiated to offer such functionality. Some of these can be thought of as "enhancing" the information model available at every interface. As an example, metric monitoring, summarisation and logging can enhance raw data such as transmitted and discarded octets across an interface to provide throughput and error rates, QoS alarms and historical trend data over time. This information can be provided by simply instantiating the relevant support managed objects and initiating the relevant functions.

An accusation often directed to the TMN is the lack of precise computational specifications complementing the GDMO interface definitions. It should be mentioned that according to the ITU-T, the TMN is first and foremost a communications concept. As such, the lack of such specifications is deliberate; specifications are concerned with what is available at a management interface rather than how this functionality is provided internally. There has already been a lot of research work towards formalising GDMO behaviour, as already mentioned in section 2.2.3, while ITU-T is considering languages such as Z [Spiv89] and SDL [Z100]. Even in this case though, additional computational intelligence will be required to support, for example, alarm correlation functions, information conversion functions etc.

Another limitation of the TMN has to do with the inherent asymmetry of the manager-agent model adopted in its information architecture, especially for peer-to-peer interactions. It is true that this model can be limiting if the underlying engineering concepts separate completely the manager and agent aspects. In flexible object-oriented support environments such as the one proposed in Chapter 4, this separation is not strong at all: a managed object may as well act as a managing object at the same time.

In summary, the TMN is a hierarchically-structured environment of co-operating distributed applications, each containing a managed object cluster or ensemble in the form of a management information tree. The unit of distribution is the TMN physical block (i.e. OS), while there exist typically more than one OSs in every TMN layer in peer-to-peer or top-down hierarchical relationships. The units of re-usability can be the managed object class, the managed object cluster that models a management service component, the OS physical block and finally the management service realised by a set of co-operating OSs.

2.5 Summary

2.5.1 Research Contribution

In this chapter we have presented first a detailed overview of OSI-SM and TMN. This has been largely based on the relevant ITU-T recommendations but the author has presented those through his own point of view, explaining the reasons behind the various architectural decisions and clarifying a number of issues. As such, there are aspects of this presentation that constitute a research contribution. The relevant contributions in the introductory sections 2.2 and 2.3 are the following.

- The association of the requirements in each of the FCAPS functional areas with the systems management functions in section 2.2.1.
- The discussion on the use of polymorphism across a management interface through generic MOCs and its relationship to allomorphism in section 2.2.3.
- The taxonomy of the SMFs according to their functionality in section 2.2.5.
- The positioning of the TMN in the B-ISDN reference model and the discussion of the different aspects of control and management in section 2.3.1.
- The association of the TMN requirements with those aspects of OSI-SM that justify its choice as the supporting base technology in section 2.3.2.
- The discussion on the relationship and aspects of the intra-layer and inter-layer TMN reference points and the proposed taxonomy in section 2.3.2.3.
- The description of the functionality and relationships of the various functional components that constitute an OSF in section 2.3.2.5.

The third part of this chapter (section 2.4) proposed a number of modifications and extensions to the TMN model and architecture and constitutes the major research contribution of this chapter. The specific research contributions are the following.

- The introduction of the OSI directory as the technology to support TMN distribution and discovery services. This includes the issues behind a relevant directory information model and the exact mechanism for name resolution and location transparency described in section 2.4.1.

- The discussion on the aspects of adaptation in section 2.4.2.1. In particular, the discussion on the aspects of generic adaptation and the conclusion that the latter typically results in Q_x interfaces, necessitating an additional mediation function which partially out-weights the relevant advantages.

- The discussion of the aspects of Q_x interfaces and the taxonomy of the mediation functions into three generic classes with particular characteristics in section 2.4.2.2. In addition, the proposal that Q_x interfaces should not be standardised and the subsequent removal of the MF from the functional architecture.

- The discussion on the relevant aspects of the workstation functions and the F interface in section 2.4.3. The suggestion that the F interface can be derived from Q_3, and the subsequent proposal that the F interface should not be standardised. This leads to the replacement of the WSF by the WS-OSF function block in the functional architecture.

- The discussion on the possible approaches for integrating the OSI Directory in the TMN functional architecture in section 2.4.4.

- The simplification of the TMN functional architecture depicted in Figure 2-22.

- The simplification of the internal structure of the OSF function block in Figure 2-23.

- The final discussion on the TMN framework as a whole in section 2.4.5 which, among other, explains the exact nature of the TMN Q_3 and X interfaces.

Since most of the research work described in this thesis has been conducted in a collaborative research project environment, it has inevitably involved other researchers. We clarify here which of the research work in this chapter has been performed in a collaborative fashion and separate the credits between the author and the researchers involved.

- The identification of the different aspects of control and management have been based on lengthy discussions and exchange of ideas with D. Griffin, formerly with ICS, Crete, Greece and currently with UCL.

- The idea that the F interface should not be standardised and the introduction of the WS-OSF function block was conceived in collaboration with D. Griffin. The credit on the concept is equal while the author worked out the precise architectural details presented in section 2.4.3 and documented the relevant concept in [Pav96a] and [Pav96d].

- The use of the OSI Directory in the TMN in order to support distribution and discovery services was worked out in collaboration with C. Stathopoulos and D. Griffin of ICS, Crete, Greece. In addition, the above group of people had discussions with M. Tschichholtz and A. Dittrich of GMD Fokus who masterminded the PREPARE approach. The details of the directory information model and the access principles fir location transparency were devised by the author and C. Stathopoulos. Regarding the relevant software design and implementation, C. Stathopoulos implemented the directory access aspects while the author integrated it in OSIMIS and devised the naming scheme for the SMAE directory objects. These aspects are described in more detail in Chapter 3.

It should be also mentioned that the contribution to the Shared Management Knowledge ITU-T recommendation [X750] was put together mainly by A. Dittrich of GMD Fokus, unifying the ICM and PREPARE approaches. The author and C. Stathopoulos commented on the various drafts.

All the other research contributions presented in this chapter are the author's alone.

2.5.2 Towards the Realisation of the TMN

Having simplified the TMN architectural framework, the key objective of this thesis is to demonstrate that it can be mapped onto object-oriented programming environments through platform abstractions similar to those of distributed object-oriented frameworks. The relevant realisation should retain the presented powerful characteristics but it should be also performant, efficient in terms of required computing resources, it should scale and it should support the rapid development of TMN applications.

Based on the analysis in this chapter and the TMN characteristics as depicted in Figure 2-22 and Figure 2-23, the relevant software environment should support the following:

- it should support the development of TMN applications in the agent role of the OSI-SM model, i.e. NEFs and QAFs;
- it should support the development of TMN applications in manager roles, i.e. WS-OSFs;
- it should support the development of TMN applications in hybrid manager-agent roles, i.e. OSFs; in addition
  - ◊ the manager-agent roles should not be separated in engineering terms so that peer-to-peer interactions are supported in a natural fashion;

- it should support distribution and discovery aspects through the OSI directory; and
- it should support secure management information exchanges, especially for inter-domain interactions across the X interface.

A key aspect for all those objectives is the realisation of the Q_3 / X protocol stack and its encapsulation in object-oriented infrastructure in a "harness and hide" fashion. Managed objects should be "first class citizens" of the resulting software infrastructure based on the OSI-SM GDMO information modelling framework. Managing objects should be able to access those in a natural fashion, disregarding physical distribution and being shielded from protocol aspects. Finally, the powerful information access aspects of the OSI-SM/TMN technology, i.e. scoping, filtering and fine-grain event reporting, should be available in a natural, easy-to-use fashion.

While WS-OSs interoperate with OSs based on Q_3 interfaces, it may be necessary to realise GUIs in scripting languages with built-in relevant support such as Tcl/Tk [Oust94] and Java [Sun96]. This implies that a string version of an access API might be also desirable. Such an API essentially implements a string-based CMIS service and could be mapped onto protocols such as HTTP in order to drive WWW displays, as discussed in section 2.4.3.

The issues behind the object-oriented software realisation of the TMN framework are examined in detail in Chapter 3.

3. Mapping the OSI-SM / TMN Model Onto Object-Oriented Programming Environments

3.1 Introduction

Chapter 3 of this thesis proposes a novel approach for the realisation of the OSI-SM/TMN framework, based on object-oriented software platforms. While the TMN is object-oriented in information specification terms, it is a communications framework and, as such, it does not address software realisation aspects. The richness and complexity of the overall framework in conjunction to the fact that non object-oriented approaches were initially adopted for its realisation, resulted in doubts about its implementability, performance and eventual deployment.

In this chapter we demonstrate how the inherent object-oriented aspects of the OSI-SM/TMN framework can be exploited through an object-oriented realisation model that hides protocol aspects through abstractions similar to those of emerging distributed systems frameworks. The resulting environment is an easy to use object-oriented distributed software platform that enables the rapid development and deployment of TMN systems. The software architecture of the proposed environment is presented while its power, expressiveness, usability and similarity to recently emerging distributed object frameworks is demonstrated through examples. The environment in which the relevant concepts and abstractions were validated is the OSIMIS TMN platform which predated similar products by some years and influenced a number of subsequent commercial developments.

Having demonstrated the mapping of the abstract OSI-SM/TMN framework to object-oriented programming environments in the form of an object-oriented distributed software platform, we subsequently demonstrate that the resulting framework has good performance characteristics. We demonstrate in particular that the main performance cost is due to the Q_3 protocol stack rather than the proposed application framework. This is particularly important since we show in Chapter 4 that it is possible to retain the TMN application aspects over a distributed object framework such as OMG CORBA.

Chapter 3: Mapping the OSI-SM /TMN Model Onto Object-Oriented Programming Environments

This chapter is organised as a "super-chapter", in a similar fashion to chapters 2 and 4 of this thesis. Related research work is presented in the various sub-sections before the author's research work, in a similar style to the rest of the thesis.

Section 3.2 presents first an introduction to object-oriented software systems and subsequently identifies a number of key properties of object-oriented distributed software frameworks against which the proposed OSI-SM/TMN realisation framework will be measured.

Section 3.3 presents key issues in realising the protocol part of the Q_3 interface, discusses possible policies for the relevant API and investigates alternative mappings over different, lightweight transport mechanisms. Section 3.4 discusses issues behind object-oriented ASN.1 manipulation, which is key to any OSI upper layer infrastructure and an essential ingredient of the proposed TMN application framework.

Section 3.5 discusses the manager or client mappings of the proposed TMN application framework. Two approaches are presented, one modelling whole remote agents and another one modelling individual managed objects. The latter includes a manager mapping of GDMO to O-O programming languages. A mapping to the Tcl/Tk interpreted scripting language is also presented, being suitable for the rapid realisation of TMN WS-OS applications.

Section 3.6 discusses the agent or server mappings of the proposed TMN application framework. It proposes an agent mapping of GDMO to O-O programming languages, discusses interaction models between managed objects and associated resources, presents realisation aspects of the OSI-SM SMFs and shows that the perceived "difficult" aspects of the OSI-SM/TMN framework, i.e. scoping, filtering, event reporting and logging, are in fact easy to realise.

Section 3.7 discusses aspects of synchronous "remote procedure call" and asynchronous "message passing" paradigms, which are both supported in the proposed environment. Section 3.8 presents a performance analysis and evaluation in terms of response times, application sizes and the amount of management traffic incurred.

Section 3.9 examines the proposed framework against the desired properties of object-oriented distributed frameworks identified in section 3.2. It also shows how the functional decomposition of the TMN OS presented in Chapter 2 is mapped onto the proposed object-oriented realisation framework. Since the ultimate validation of the latter was accomplished through research and development work based on the proposed environment, such work is presented in Appendix A.

Finally, section 3.10 highlights the research contributions in this chapter.

3.2 Object-Oriented Distributed Systems

One of the key contributions of this thesis is that it demonstrates how the OSI-SM / TMN model can be mapped onto object-oriented programming environments using abstractions similar to those of emerging object-oriented distribution frameworks. It is thus important to define first the terms *object-oriented programming environment* and *object-oriented distribution framework.*

3.2.1 Object-Oriented Development Principles

Object-orientation has been a cultural achievement of software engineering in the mid and late eighties. It proposes a new approach for specifying, designing and implementing software systems which takes further the structured approach of the past and achieves new levels of software reusability, extensibility and genericity. Object-oriented concepts and principles have already been mentioned when describing the OSI-SM information model in section 2.2.3 of Chapter 2. Here we attempt a more systematic definition.

In traditional or *structured* software engineering, programs comprise data structures and logic which are loosely coupled. Designers and programmers think of their programs in terms of the required logic first and add data structures later in order to support the needs of that logic. The relevant data structures are globally available and accessible by different program procedures which manipulate them. Program logic is developed by "stepwise refinement" [Wirth71] while the basic building block is the procedure. A typical programming language that supports this paradigm is Pascal [Wirth75].

An evolution of the structured approach has led to the *modular* paradigm. A module implements an abstraction that becomes the basic building block of complex programs. A module has well defined functionality, e.g. it implements an abstract data type such as a linked list, and comprises both procedures and data, in a similar fashion to a modular program. Data is hidden inside the module so that it becomes invisible to procedures of other modules. This principle is known as data-hiding or encapsulation and guarantees the internal consistency and integrity of the module. The module's functionality is made available to other modules through well-defined entry points, implemented as "public" procedure calls.

A module can be thought as some form of object since it supports encapsulation. In fact, some refer to this approach as *object-based.* A key drawback is that a module may only have one "instance" since it contains a single copy of the private data. In addition, a module's functionality cannot be modified or extended without having access to its source code. As such, this approach supports only limited reusability and extensibility. Programming languages that support this paradigm are C [Kern78], Modula [Wirth82] and Ada. It should be noted that the C programming language supports this paradigm implicitly only, since there are no explicit language constructs to support modules.

The evolution of the modular framework has led to the *object-oriented* paradigm. An object is similar to a module since it contains procedures and data, but the two are tightly coupled and constitute an object type or *object class*. The procedures of an object are known as *methods* and its data as *variables*. Many *instances* of the same class may exist at any time, with their own copies of instance variables. Access to the latter is allowed only through an object's methods. Some of those are *private* and cannot be accessed from outside. *Public* methods can be accessed by other objects in order to perform certain functions. A method may change the state of an object, operate on some of its variables or act on other objects. In an object-oriented system, all interactions among object instances take place through method calls or *messages*. This paradigm provides better support for software reusability and system integrity.

The concept of an object is taken further through inheritance and polymorphism. *Inheritance* allows a new class, called a *subclass*, to be an extension, modification or even restriction of the original class, called the *superclass*. A subclass may include additional methods not present in the superclass (extension), may override existing ones (modification) or may even prevent existing ones from exercising their functionality (restriction). These features can be supported without access to the source code of a superclass and provide excellent support for software re-usability and extensibility.

Polymorphism is an intriguing characteristic which enables one to treat instances of derived classes as instances of a generic superclass (from the Greek words *poly:* multi and *morphe*: shape or form). A typical example demonstrating the use of polymorphism is that of a window manager object which treats displayed objects in the same way, regardless of their particular specialisation e.g. word processor window, task-bar, pointer etc. The window manager is programmed to interact with instances of a generic class, e.g. displayableObject, which could be moved, resized, iconified, brought forward or backward and so on. It can then interact with instances of particular specialisations of that class and trigger associated behaviour without even knowing what the relevant classes are.

Programming languages that support the object-oriented paradigm are Smalltalk [Gold83], C++ [Strau86], Objective C [Cox86], Eiffel [Meyer88] and more recently Java [Sun96]. The most popular of those is C++ because of its compatibility with C and the fact it is highly efficient.

Polymorphism in C++ is supported by *virtual* methods which may be redefined in derived classes. A call to virtual method results in triggering the *leaf-most* method implementation in the inheritance hierarchy of that instance, despite the fact that the caller "sees" the latter as an instance of a generic superclass. This feature achieves polymorphic behaviour.

Object-oriented programming should be based on a sound object-oriented design. The latter breaks away from the structured and even modular design practices and proposes a new approach to the decomposition of complex systems. There exist a number of books addressing object-oriented decomposition methodologies. [Booch91] and [Rumb91] are the classical references, discussing both issues of object-oriented decomposition and proposing modelling techniques for documenting an object-oriented design, the Object-Oriented Design (OOD) and the Object Modelling Technique (OMT) respectively. [Cox86] and [Meyer88] address mainly O-O programming languages, Objective C and Eiffel respectively, but they also contain useful material on object-oriented design. This thesis uses OMT, C++ class specifications and object instance diagrams to demonstrate aspects of object-oriented design.

The OSIMIS platform, which is the environment in which the ideas presented in this thesis have been validated, was designed using object-oriented design principles and making extensive use of concepts such as inheritance and polymorphism. The goal behind the design was to allow reusability, extensibility and access to sophisticated features through simple-to-use object-oriented APIs. The approach was baptised *harness-and-hide* [Pav94b]. C++ [Strau86] was chosen as the programming language, the reasons being at the time (1989) compatibility with C, ubiquity, strong type checking and performance.

3.2.2 Object-Oriented Distribution Frameworks

Distributed systems have been addressed since the early eighties by the research community and have become a reality since the mid to late eighties through the advent of local area networks and inexpensive workstations and personal computers. Distributed systems exhibit component remoteness, component concurrency, lack of precisely determinable global state and potential for partial failures. On the other hand, they offer potential advantages in availability, performance, dependability and cost optimisation resulting from distribution. A key issue in distributed systems is masking the heterogeneity of the hardware, operating systems and programming languages used to build them. [Coul88] addresses the concepts and design of distributed systems in detail while [Kram94] provides a concise introduction to the relevant issues. It should be noted that most of the literature on distributed systems assumes silently a highly reliable and "fast" local area network as the supporting communications infrastructure. This is not the case with a TMN which can be distributed over a wide area network, with parts of it communicating over slower, less reliable links.

Since the early days of research in distributed systems, a key requirement has been the extension of programming languages with constructs to support distributed computation. There exist two different paradigms for those extensions: unidirectional asynchronous message passing or bi-directional synchronous Remote Procedure Call (RPC), the latter having semantics similar to a local procedure call. There exist both differences and complementary aspects in the two approaches which are discussed in more detail in section 3.7 of this thesis.

The RPC paradigm is described in the seminal [Birr84]. Since its inception, a number of distribution frameworks based on it appeared, providing support for the development and deployment of distributed systems. Sun Microsystems' RPC [Sun88] comes bundled with their SunOS and Solaris operating systems and has been widely used. The UCL RPC environment is described in [Wilb87] and included a binding service to support location transparency, introducing aspects of an elementary platform. The ANSA platform [ANSA89a] introduced the concept of trading and was more than an RPC environment, influencing the development of the whole ODP framework [ODP]. The OSF DCE was a industrial approach, bearing more similarities to the first two systems than to ANSA. OMG CORBA [CORBA] is another, more recent, industrial approach, embracing for the first time true object-orientation. The author has experimented with all of those frameworks apart from DCE. The ANSA, DCE and CORBA frameworks are examined in more detail in Chapter 4 of this thesis.

Since we will later need to evaluate the proposed OSI-SM/TMN realisation approach against the properties of *object-oriented distribution frameworks,* it is important to define what these properties are. In an ideal distribution framework, one could take a non-distributed object-oriented program, derive abstract specifications for the object interfaces, produce distributed "stub" objects through relevant tools and re-use most of the existing implementation to fill-in the stub objects with behaviour. After this reverse-engineering process, the system could be deployed in a distributed fashion.

The observant and cognisant reader may remark that this is what Java's [Sun96] Remote Method Invocation (RMI) mechanism tries to achieve, without the need for an intermediate step of abstract interface specification. The latter is exactly the point: the Java RMI assumes an homogeneous environment where all distributed objects are programmed in Java. A distribution framework should mask the heterogeneity of components, acting as a unifying "glue". It should also *allow* for it in the first place and be able to *cope* with it. This is exactly what the Java RMI does not do and this is why it is not considered as a distribution framework in this thesis. It should be noted that distributed objects could be programmed in Java in any other framework.

Distributed objects need to be specified in an abstract language, which should be programming language independent. That language should be object-oriented, supporting inheritance and polymorphism, since the latter are key properties of object oriented systems as explained. Distributed objects could be developed in different programming languages through multiple language mappings. These should include mappings to object-oriented languages, which would be most natural given the object-oriented nature of the abstract language itself. In complex distributed systems there is a need for generic applications which can operate without statically built-in knowledge of the objects they access. Such applications need to use a *dynamic invocation* facility. Finally, the relevant environments should be easy to use by hiding communication details. They should also be performant and scaleable in order to encourage distribution.

ODP [ODP], which is examined in more detail in Chapter 4, identifies a number of properties of distributed systems. *Openness* addresses both software portability through standard APIs and requires interoperability through agreed communications protocols. *Distribution transparencies* mask the details of the mechanisms used to overcome distribution problems. These include among other *access transparency*, which masks differences in data representations and remote execution, and *location transparency*, which masks the location of a distributed component providing a service.

We have thus identified the following key properties of object-oriented distribution frameworks:

- an abstract, object-oriented specification language that supports inheritance and polymorphism
- mappings of the abstract language to object-oriented and also procedural/modular programming languages
- user friendly APIs that hide communication and protocol details
- dynamic access facilities that obviate the need for static (i.e. pre-compiled) knowledge of object specifications in client applications
- good performance and scalability so that distribution is encouraged and exploited
- openness in terms of both standard APIs and communication protocols

- distribution transparencies, and in particular access and location

The rest of this chapter explains the issues behind a C++-based software architecture that realises the OSI-SM / TMN model in a distributed object-oriented framework fashion. We will examine the proposed framework against the above properties at the end of this chapter, in section 3.9.1. We will also examine ANSA, the OSF DCE and OMG CORBA against the properties set above in Chapter 4.

3.3 Issues in Realising the Protocol Part of the Q_3 Interface

In this section, we consider issues associated to the mapping of the OSI-SM/TMN Common Management Information Service and protocol (CMIS/P) [X710][X711] onto object-oriented environments through suitable APIs. A brief introduction to CMIS/P has already been given in section 2.1.4 of Chapter 2. We will start this section by examining CMIS/P and the supporting OSI protocol stack in more detail. We will then discuss relevant research work and will present our approach, discussing also alternative design possibilities.

3.3.1 The Q_3 Protocol Stack

As discussed in section 2.2.1 of Chapter 2, TMN traffic may use the telecommunications network being managed. In addition, parts of the TMN operate in other networks attached to the telecommunications network. This implies that the TMN Q_3 protocols need to operate over a number of diverse lower layer data network technologies, spanning from X.25 and the Signalling System No. 7 (SS7) to the Internet TCP/IP, which is rapidly becoming the dominant data network technology. The lower layer stack profile for the Q_3 interface is specified in [Q811]. This comprises a number of sub-profiles as depicted in Figure 3-1.[1]

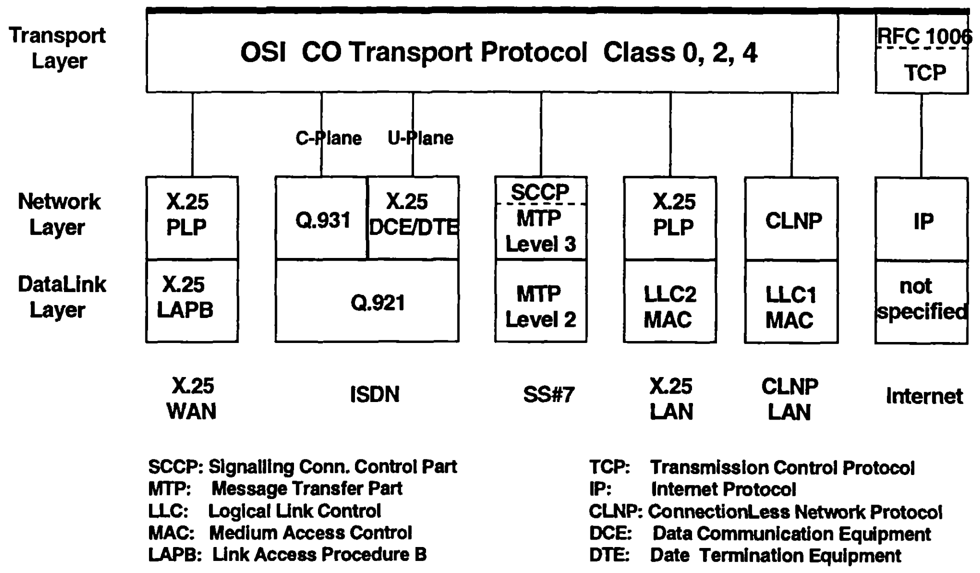

Transport Layer
OSI CO Transport Protocol Class 0, 2, 4
RFC 1006
TCP
C-Plane
U-Plane
Network Layer
DataLink Layer
X.25 PLP
X.25 LAPB
Q.931
X.25 DCE/DTE
Q.921
SCCP
MTP Level 3
MTP Level 2
X.25 PLP
LLC2 MAC
CLNP
LLC1 MAC
IP
not specified
X.25 WAN
ISDN
SS#7
X.25 LAN
CLNP LAN
Internet
SCCP: Signalling Conn. Control Part
MTP: Message Transfer Part
LLC: Logical Link Control
MAC: Medium Access Control
LAPB: Link Access Procedure B
TCP: Transmission Control Protocol
IP: Internet Protocol
CLNP: ConnectionLess Network Protocol
DCE: Data Communication Equipment
DTE: Date Termination Equipment

Figure 3-1 Lower Layer Protocol Profile for the Q3 Interface (from [Q811])

[1] Familiarity is assumed with the OSI 7 layer reference model [X200] and data network technologies in general. A good introduction can be found in [Tanen96].

The X.25 wide area profile, the ISDN and the SS7 profiles are the natural candidates when managing X.25, ISDN and SS7 networks respectively. It should be noted that the performance of the ISDN and SS7 signalling protocols tends to be very high in a wide area, which may not be the case with traditional data network technologies such as X.25 and the Internet TCP/IP. When managing plain transmission networks such as SDH and SONET, any of the previous technologies may be used over the "embedded communications channel". In the case of the B-ISDN which will be based on ATM technology, additional mappings will be defined over both the relevant signalling [Q2931] and user planes. Finally, for parts of the TMN operating in local area networks, it is possible to run either X.25 or the OSI ConnectionLess Network Protocol (CLNP).

In all the above combinations of network and data link protocols, the OSI Connection-Oriented Transport Protocol (COTP) provides the end-to-end Connection-Oriented Transport Service (COTS). The use of COTP class 0, 2 or 4 depends on the reliability characteristics of the underlying network service. Interoperability between subnetworks of different network technologies can be achieved either through network layer relaying, which involves protocol conversion, or through transport service bridging. We have discussed briefly the issues of protocol conversion and service bridging in section 2.3.2.2 of Chapter 2, while explaining aspects of mediation functions.

All the above technologies are pure OSI ones. Over the last years though, the Internet TCP/IP has undoubtedly become the dominant data network technology. As such, the ITU-T recognised the need to support a TCP/IP-based profile for the Q_3 interface. This can be done by treating TCP as a reliable network protocol, in a similar fashion to X.25, and operating over it a convergence protocol that provides the OSI COTS. The key difference between the COTS and the service offered by TCP is that the former is packet-oriented while the latter is stream-oriented. As such, the convergence protocol consists of two parts: a small "packetisation" protocol over TCP, which makes it appear as an OSI network protocol; and the OSI TP class 0 over the packetisation protocol that offers the COTS. This approach was standardised through the RFC 1006 [Rose87]. The intention is to enable OSI upper layer protocols and applications to operate over the Internet lower layer protocols. A good discussion of the relevant issues can be found in [Rose90].

Using this approach, the Q_3 upper layer protocols may operate over TCP/IP in a completely transparent fashion. Note though that this approach is *different* and not interoperable to the CMOT [Besa89] approach which will be discussed in section 3.3.2.4. Interoperability between Q_3 stacks based on TCP/IP and Q_3 stacks based on any of the other OSI lower layer technologies can take place through transport service bridging [Rose90]. The transport service bridge should run on the network node that interconnects the subnetworks of the two different technologies, e.g. on the node that connects a TCP/IP local area network to the SS7 telecommunication network.

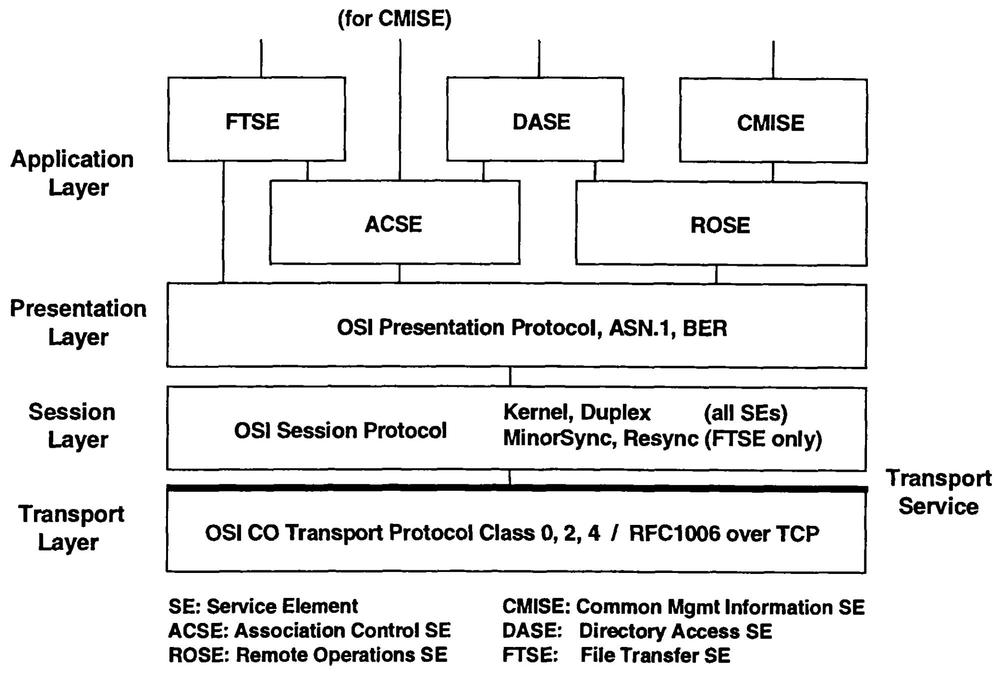

(for CMISE)
FTSE
DASE
CMISE
Application Layer
ACSE
ROSE
Presentation Layer
OSI Presentation Protocol, ASN.1, BER
Session Layer
OSI Session Protocol
Kernel, Duplex (all SEs)
MinorSync, Resync (FTSE only)
Transport Service
Transport Layer
OSI CO Transport Protocol Class 0, 2, 4 / RFC1006 over TCP
SE: Service Element
ACSE: Association Control SE
ROSE: Remote Operations SE
CMISE: Common Mgmt Information SE
DASE: Directory Access SE
FTSE: File Transfer SE

Figure 3-2 Upper Layer Protocol Profile for the Q3 Interface

While the lower layer Q3 profile may vary, the upper layer profile is always the same as shown in Figure 3-2. The main Application Service Elements (ASEs) that are part of the Q_3 interface are CMISE [X710] and DASE [X511], while FTSE [FTAM] may be also used in the future. The service provided at the Transport Service Access Point (TSAP) is a reliable, packet-based service that does not support graceful connection release. The OSI Session Protocol adds graceful connection release, half-duplex exchanges through token management, dialogue control through checkpointing and synchronisation, activity management and exception reporting. Both CMISE and DASE need none of the sophisticated functionality of the session layer and use only the basic kernel and duplex services. The File Transfer SE needs also the Minor Synchronisation and Re-synchronisation services.

While the Session Service Access Point (SSAP) supports data exchanges with no structure, the Presentation Protocol adds structure to the data through the Abstract Syntax Notation One (ASN.1) [X208] language. ASN.1 implements an *abstract syntax* whose data structures need to be converted to byte streams and transmitted across the network and vice-versa. This functionality is provided by various sets of Encoding Rules (ER) that implement different *transfer syntaxes*. The mapping of an abstract syntax to a transfer syntax is termed a *presentation context*. The various ASEs may use different presentation contexts which are negotiated at connection establishment time. The presentation layer keeps track of the presentation contexts and provides syntax matching functions that serialise and de-serialise the relevant data structures. The Q_3 interface specification [Q3] suggests the use of the Basic Encoding Rules (BER) [X209] as the transfer syntax.

The application layer structure consists of a number of layered ASEs. ACSE [X217] manages application layer connections which are termed *associations*. It provides a combined interface to the PSAP and SSAP connection management services but adds also Application Entity Title (AET) parameters for the calling and called parties. A AET in its complete form is the directory name of the application as explained in section 2.3.1 of Chapter 2, e.g. {c=GB, o=UCL, ou=CS, cn=ATM-NM-OS}. A simpler form for an AET is the value of the application process relative name, e.g. ATM-NM-OS. OSI applications use the ACSE services either directly, e.g. for establishing CMISE associations, or indirectly through other ASEs, e.g. for establishing DASE and FTSE associations (see Figure 3-2).

ROSE [X719] realises the OSI mechanism for building distributed applications based on a *request/response* paradigm. Though asynchronous in nature, it can also support synchronous Remote Procedure Call (RPC) semantics [Birr84], which is what many distributed applications are built on. Both CMISE [X710] and DASE [X511] use ROSE to implement management information and directory access operations respectively. We are going to discuss ROSE and CMISE in more detail while addressing their realisation, since they are the main components of the upper layer Q_3 profile. The picture of the latter is completed by FTSE for file transfer [FTAM], which uses directly the presentation layer services.

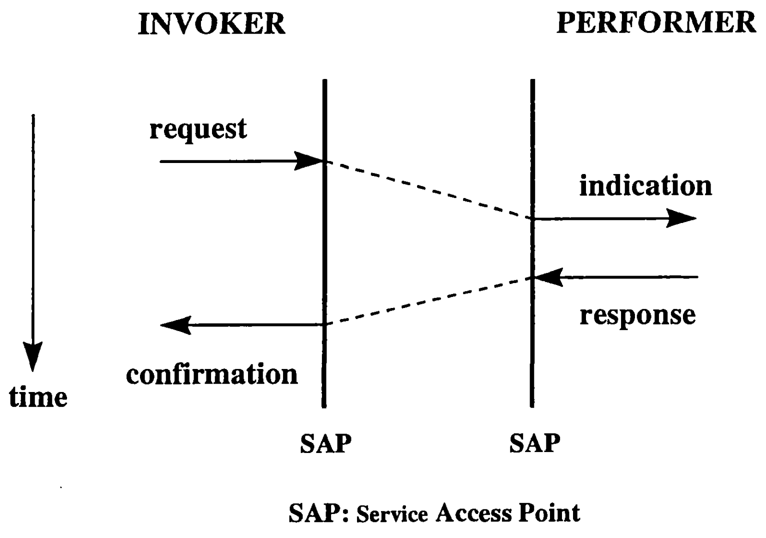


Figure 3-3 OSI Invoker and Performer Interactions

In general, interactions between adjacent OSI layers or application layer ASEs take place at Service Access Points (SAPs), which offer the services of the subordinate layer or ASE. Exchanges between peer entities across the network follow a request-indication cycle, followed by a response-confirmation cycle for confirmed exchanges. The requesting user of a service is termed an *invoker* while the accepting user is termed a *performer*. This interaction model is depicted in Figure 3-3.

3.3.2 Issues in Realising the Upper Layer Part of Q_3

3.3.2.1 General Issues in Realising Upper Layer Infrastructures

Realising an upper layer Q_3 stack profile requires lower layer protocol infrastructure. Lower layers based on TCP/IP, X.25 and TP4/CLNP exist for most multi-purpose operating systems such as UNIX and WindowsNT. It should be noted that TCP/IP support comes typically bundled at no additional cost, while one has to pay extra for OSI lower layer protocols. Upper layer infrastructure, including Q_3 support for CMISE at least, can be bought today from many vendors of OSI and TMN systems. Back in the mid eighties there were no products available while the provision of efficient and reusable upper layer OSI stack infrastructure was a research issue. A major research effort in realising OSI upper layer protocols and applications and validating the relevant specifications has been the ISO Development Environment (ISODE) [ISODE][Rose90]. This provided support for the upper layer stack including ACSE, ROSE, DASE and FTSE and was used as the basis for the OSIMIS platform.

When designing software abstractions for ASEs based on a particular upper layer stack, one has the freedom to be different from the supporting infrastructure since the latter can be hidden using encapsulation. For example, ISODE is based on the structured or modular paradigm with APIs in the C programming language [Kern78] while OSIMIS is based on the object-oriented paradigm, with APIs in C++ [Strau86][Ellis91]. The ISODE APIs are not at all visible when using OSIMIS since they are encapsulated in the OSIMIS infrastructure. An important aspect related to the supporting environment though is that it might not be possible to hide all its aspects completely. This concerns in particular ASN.1 manipulation, as explained next.

Presentation layer support comes typically through ASN.1 compilers which produce concrete programming language representations for the relevant types. They also produce relevant logic for converting those representations to and from a generic representation that is understood by the presentation layer; the latter converts those to and from byte streams according to the relevant transfer syntax. An ASN.1 compiler with C mappings produces C data structures and separate encode, decode and print functions while a compiler with C++ mappings produces C++ classes with relevant encode, decode and print behaviour. A good discussion on ASN.1 compilers and relevant issues can be found in [Neuf90].

The mapping of ASN.1 to a programming language realises a ASN.1 API. This can be modified to reflect the taste of the designer of an application layer infrastructure. For example, the ISODE ASN.1 API is procedural while the author designed and implemented an additional "wrap-up" compiler that produces encapsulating C++ classes, which will be described in section 3.4. The functionality of the latter though depends on the conventions behind the encapsulated C structures produced by the native ISODE ASN.1 compiler. This observation can be generalised as follows: the designer of application layer infrastructure is somewhat "restricted" by the native ASN.1 API. This restriction can be completely removed only if a new ASN.1 compiler and associated API is designed and implemented.

3.3.2.2 Related Work on CMISE APIs

Before we move on to discuss issues behind the realisation of CMISE and relevant APIs, let's look at related work in this area and position the work presented here. The author's design and implementation of an ISODE-based CMISE that constituted the initial component of OSIMIS dates back to 1989. At that time, there was no related research work or a similar commercial product. In fact, the OSIMIS CMISE implementation served for some time as the only available reference implementation and was subsequently used as the basis for a number of products. The OSIMIS CMISE design decisions and relevant abstractions are described in [Pav93a], a tutorial on "Implementing OSI Management". The relevant API is documented in [Pav93b], the OSIMIS-3.0 manual. Brief descriptions are also given in [Pav95a] and [Pav96b] which describe the OSIMIS platform as a whole.

The only other work in the literature that discusses CMISE realisation issues is [Dens91], [XMP] and [Chat97]. The first one [Dens91] discusses the realisation of DEC's CMIS services while the latter two present work of standards bodies. The first of those is X/Open's Management Protocols API specification (XMP) [XMP], released in 1992. The second is a recent attempt by the NMF to provide object-oriented TMN APIs [Chat97], including a CMIS API known as CMIS/C++.

[Dens91] describes DEC's approach for a CMIS API in their Enterprise Management Architecture (EMA) [Strut94]. The particularly interesting aspect of their approach is that their API is a generic protocol-independent one, which can be mapped onto particular protocols through different Access Modules (AMs) [Struct89]. The interface consists of a single procedure which takes as parameters the *verb* or directive, the *in_entity* or object to access, the *attributes* for get and set directives, the *in_q* for additional qualifiers (e.g. access control) and the *in_p* for additional input arguments (e.g. action argument). The *out_p* contains the results/errors while the *out_entity* parameter contains information on the object(s) on which the directive was performed e.g. the class and name of the object. Scope and filter information are part of the in_entity parameter but only single level scoping is possible while not all the aspects of CMIS filtering are possible. This interface was obviously designed before CMIS and cannot cope fully with the richness of the latter. On the other hand, it is an interesting attempt on a generic, polymorphic, dynamic invocation interface that can be mapped onto different protocols.

The X/Open XMP interface was the first attempt from a standards body to standardise a CMIS API. The intention behind such an API is to separate OSI-SM/TMN applications from the underlying CMIS/P protocol stack so that portability across different vendors' stacks is possible. This API was first introduced in 1992 and has similarities to the OSIMIS one which had been publicly available since 1990. This is a procedural API in the C language. Every CMIS request and response primitive maps to a corresponding procedure that can be called asynchronously e.g. Get-req() and Get-rsp(). A Receive() procedure needs to be called to receive the result. A synchronous call model with RPC semantics is also supported e.g. Get(). Management "sessions" need to be established before sending and receiving messages through the Bind() call while they may be terminated through the Unbind() call. Finally, automatic name to address resolution is provided that maps application names to addresses.

While all this design makes sense and is in fact extremely similar to that of the OSIMIS CMIS, it has two serious drawbacks. First, the API tries to cater both for CMIS and SNMP and this creates unnecessary complexity. CMIS and SNMP follow very different philosophies as explained in [Pav94d][Pav97a] and also in [Geri94] and elsewhere. As such, there is no tangible benefit from unifying their access APIs while additional complexity is introduced for dealing with the different object models, parameters to common primitives etc. A second and more important drawback concerns the use of the associated X/Open ASN.1 API [XOM]. This takes an object-oriented view of structural information but does not incorporate the characteristics of object-oriented systems as explained in section 3.2.1. In particular, the functions for manipulating objects are separate from the definitions of those objects and there is no notion of encapsulating or hiding the information associated with objects. We could characterise both XMP and XOM as object-based instead of object-oriented. In summary, the combined XOM/XMP API is complex and daunting to use.

The object-based nature and complexity of XOM/XMP has led the NMF to define recently the NMF/C++ API, which comprises ASN.1, CMIS and GDMO APIs [Chat97]. This work is very much related to the work described in this chapter and has been produced by a group of experts over a 2-3 year period. The author has initially participated in that group and the OSIMIS APIs have been one of the relevant inputs. The CMIS API is known as CMIS/C++. This offers a set of C++ classes for modelling the CMISE and ACSE primitives and their parameters in an asynchronous fashion only. It also offers a set of objects for referring to outstanding operations (invocation handles) and two different mechanisms in order to receive operation indications and confirmations: a callback facility through a callback class and a queue facility through a queue class. A "convenience" API is also available for automatic association management but applications can avoid using this and can take explicit control of association establishment and release.

3.3.2.3 Issues in Realising CMISE Over ROSE

The OSIMIS CMISE implementation is based on the ISODE environment and uses the relevant ASN.1 compiler known as *pepsy* and the associated ASN.1 API. An early implementation based on the still evolving CMIS/P ISO documents was produced by S. Walton of UCL under the auspices of the ESPRIT INCA project, in 1988. A management system for monitoring the activity of the OSI transport protocol was developed based on it, as described in [Knig89]. The CMIS/P standards [X710][X711] achieved a state of maturity in 1989 and the author re-designed and re-implemented completely CMISE in late 1989. This became the fundamental building block for the OSIMIS platform and has remained fairly stable ever since, used subsequently in a number of commercial products.

Since CMISE is based on ROSE, a ROSE implementation is necessary while ACSE is also necessary for association management. ISODE provided both ACSE and ROSE implementations. In cases where the available OSI stack provides presentation layer services only, implementing ACSE and ROSE is fairly straightforward. ACSE is essentially a wrap-up of the PSAP connection management features. ROSE implements a simple, generic request/response protocol for distributed OSI applications. It also provides a facility of operations *linked* to another operation, which can be thought as remote callbacks. This facility is used by CMISE for operations resulting in multiple replies through scoping.

ro-invoke	invokeId, linkedId,	operation,	argument
ro-result	invokeId,	operation,	result
ro-error	invokeId,	error,	parameter

Table 3-1 ROSE Primitives and Associated Parameters

Table 3-1 shows the ROSE primitives (apart from *ro-reject*) and associated parameters. Because of its asynchronous nature, a unique identifier needs to be associated with every outstanding request (invokeId). Callback invocations are linked to the initial operation through a linked identifier (linkedId) which should have the value of the original invoke identifier. The operation code, argument, result, error code and error parameter are defined by higher level protocols e.g. CMISE.

Implementing a ROSE protocol machine is not difficult. The simplest policy for an associated API is a procedural asynchronous one, with a procedure modelling each of the primitives and their parameters, e.g. RoInvoke(), RoResult(), RoError(), and a separate procedure for receiving indications and confirmations, e.g. RoWait(). This is exactly the API policy ISODE implements. A relevant design decision is if the user of ROSE will be given responsibility for the uniqueness of the invokeId parameter or if the latter will be assigned by ROSE, passing it back to the caller as a "voucher" in order to be matched against the reply and linked invocations. ISODE has decided to leave this responsibility to the caller.

The state information required by a ROSE protocol machine is very little i.e. the outstanding request and indication invokeId's for a session so that further invocations, results and errors can be checked for consistency. ROSE implementations support typically *at most once* reliability characteristics, with an operation requested exactly once and the performer keeping no state of previous invokeId's. *Exactly once* reliability characteristics are also possible, with the invoker requesting repeatedly the operation with the same invokeId until a result/error or a "duplicate operation" rejection is received. In this case, the performer needs to keep additional state of the invokeId's of operations in a session from an epoch date. ROSE supports *total* distributed operations: for any given operation, the result and all exceptions (errors and rejections) are well-defined and distinguishable. The concept of totality is important for reliable distributed systems.

m-create	invokeId, access, objClass, objName, referenceName, attrList
m-cancelGet	invokeId, getInvokeId
m-get	invokeId, access, objClass, objName, scope, filter, sync, attrIdList
m-set	invokeId, access, objClass, objName, scope, filter, sync, setReqList
m-action	invokeId, access, objClass, objName, scope, filter, sync, actionInfo
m-delete	invokeId, access, objClass, objName, scope, filter, sync
m-eventRep	invokeId, objClass, objName, eventTime, eventType, eventInfo

m-createRes	invokeId, < objClass, objName, time, attrList | error, errorInfo >
m-cancelGetRes	invokeId, < | error, errorInfo >
m-getRes	invokeId, linkId, < objClass, objName, time, getAttrList | error, errorInfo >
m-setRes	invokeId, linkId, < objClass, objName, time, setAttrList | error, errorInfo >
m-actionRes	invokeId, linkId, < objClass, objName, time, actionReply | error, errorInfo >
m-deleteRes	invokeId, linkId, < objClass, objName, time, | error, errorInfo >
m-eventRepRes	invokeId, < objClass, objName, time, eventReply | error, errorInfo >

Table 3-2 CMIS Primitives and Associated Parameters

Table 3-2 shows the CMIS request and response primitives and associated parameters. *The m-get*, *m-set*, *m-action* and *m-delete* primitives may operate on many managed objects through the *scope*, *filter* and *sync* parameters. The base object for the search is identified by the *objName* parameter. When these primitives are applied to a single object instance (i.e. without scope and sync), the optional *objClass* parameter may be used to request allomorphic behaviour. In the case of the *m-create* primitive, the objClass parameter is mandatory while objName is optional for classes with "automatic instance naming" properties. The setReqList parameter of m-set is a list of `{attrId, attrVal, modifyOperator}` tuples. The modify operator can take the values *replace*, *setToDefault*, *add* and *remove*, the latter two for multi-valued attributes [X720]. The access parameter is reserved for access control [X741] but its use has not yet been defined. The rest of the parameters are self-explanatory.

The response primitives model both result and error conditions. The relevant result parameter is passed back together with the objName, objClass and a timestamp. In case of an error, the error code is passed back together with relevant error information. CMIS/P defines a comprehensive set of errors [X711]. It also allows for object-specific errors through the processingFailure error.

Implementing a CMISE protocol machine over ROSE is not difficult, despite the fact that CMISE [X711] is a much more complex protocol than ROSE [X219]. While a ROSE protocol machine can be implemented without the need for an ASN.1 compiler due to the reduced primitive set and the simple parameter types (ASN.1 INTEGER and ANY), CMISE needs ASN.1 compiler support because of its complexity. In the case of requests and responses, the main task of a CMISE protocol machine is to assemble the API parameters, create and encode a CMISE Protocol Data Unit (PDU) and use the relevant ROSE primitive. In the case of indications and confirmations, the CMISE PDU should be decoded and the API parameters should be populated. The only state information that needs to be kept concerns outstanding m-get requests so that m-cancelGet requests are validated at source.

3.3.2.3.1 Association Management

A CMISE API should provide access to the relevant services in an efficient, flexible and easy-to-use manner. CMISE services can only be used after an association has been established through ACSE. An important design decision to make is whether the CMISE user will be given control of establishing and releasing ACSE associations or such activities will be handled transparently by the infrastructure. This decision has an impact on the API and there are three possible design decisions:

a) association management becomes an explicit part of the CMIS API; this is the approach followed in OSIMIS [Pav93b] and the NMF CMIS/C++ [Chat97];

b) Bind and Unbind facilities to management applications are part of the API but association management takes place transparently while in the bound state; this is the approach followed by XMP [XMP]; and

c) all the CMIS operations accept some form of global names, with the prefix part denoting the management application.

The third one is the most abstract. In the case of a procedural API with a procedure for each primitive, an additional API parameter is required for those CMIS primitives that do not include a name in the remote system, i.e. m-cancelGet, m-create and m-eventRep. This parameter should be the distinguished name of the target application. An important drawback of this approach is that it hides completely the relevant negotiation capabilities at association establishment, which can only take place in a pre-packaged fashion behind the API. This is fine only as far as the applications' requirements are in accordance with the pre-packaged policy. But as [Chat97] points out, "a convenience interface is only convenient if it does what you need"!

In the case of b), association options to a particular destination can be specified through the Bind() primitive. Some form of identifier is passed back from Bind() which should be used as a prefix to all the primitives. Note that this is *not* an association handle but denotes the binding with that system. Associations will be opened and closed transparently by the infrastructure thereafter. Finally, a) is the most "low level" approach but also most powerful, since it allows explicit control of associations. Connect and disconnect primitives are available, with Connect() typically returning an association handle to be used as a prefix in the other primitives.

In any of the schemes presented above, the destination can be specified through a logical application name e.g. NM-OS or the full name {c=UK, o=UCL, ou=CS, cn=NM-OS}. The latter is typically required only when crossing domain boundaries since the local domain name is known by the infrastructure. If location transparency is supported through the OSI Directory [X750] or any similar mechanism, the infrastructure will map this name to an address. If location transparency is not supported, the location name needs to be passed together with the application name e.g. NM-OS@athena or {c=UK, o=UCL, ou=CS, cn=NM-OS, cn=athena}. In this case, some form of local database is used to map this name to an OSI presentation address. The problem with this approach is twofold, as already discussed in section 2.3.1 of Chapter 2: first, there is no location transparency; and second, it is very difficult to keep those local databases consistent in a large-scale distributed system.

From the three schemes presented above, the author chose to implement a) because it offers the maximum expressive power. In addition, it models explicitly the ACSE [X217] specification and its use dictated by CMISE [X711]. The latter is an important reason since the relevant standard documents or typical textbook explanation of the OSI application layer structure could serve as reference documents on the structure and semantics of the API. The same reasons hold also for the NMF CMIS/C++ [Chat97], though the latter also offers an additional "convenience" API for automatic association management. In OSIMIS, such an API is only offered at a higher level as described in section 3.5.

The author also chose to unify the ACSE and CMISE APIs under one common API. This API supports location transparency through the OSI Directory as dictated by [X750], but it can also be used in a non-transparent fashion through a local database. In the former case, directory access takes place "underneath" the combined CMISE / ACSE API.

3.3.2.3.2 Procedural vs. Object-Oriented APIs

Another important decision regarding the API concerns the use of a structured, object-based approach or an object-oriented one. The decision here should be clear-cut: an object-oriented approach offers important advantages in terms of reusability, simplicity, easier state and memory management, etc. For example, one could model the CMISE protocol machine as an object, with methods corresponding to the relevant service primitives. This object would also encapsulate ACSE features as described above. An instance of that object would model a (remote) management interface, encapsulating the relevant binding and association information.

The X/Open XMP [XMP] has chosen an object-based as opposed to an object-oriented approach. The NMF CMIS/C++ [Chat97] follows a fully object-oriented approach; this is also the case with the OSIMIS high-level manager API known as Remote MIB (RMIB) [Pav94b] that will be described in section 3.5. On the other hand, the OSIMIS CMIS API, known as the Management Service Access Point (MSAP) API [Pav93b] follows a procedural approach and is implemented in C, in a similar fashion to the XMP one.

The main reasons for the decision not to follow an object-oriented approach, at least for the CMIS API, were political rather than technical. At that time (second half of 1989), the intention was to make the CMISE protocol machine part of the ISODE distribution. This would increase the popularity and acceptance of OSI-SM as a whole due to the wide deployment of ISODE in the research community. ISODE is written in C and follows a procedural approach throughout, so the same approach should be followed for CMISE. It should be noted that the main ISODE contributor, M. Rose, was at the time involved in the standardisation of SNMP [SNMP] and his views were pretty vitriolic regarding OSI-SM; an amusing tale of his can be found in [Rose91]. Because of his views, the OSIMIS CMISE implementation was never incorporated in ISODE, which meant that the original CMISE design and implementation *could* have been object-oriented. In fact, after the wide deployment of OSIMIS in the mid-90's, the fact that OSIMIS required the ISODE stack and its ASN.1 tools was considered by many as a liability.

The OSIMIS CMISE implementation follows an asynchronous procedural paradigm, with every request and response primitive in the Table 3-2 mapped to a separate procedure. The parameters of those primitives are mapped directly to those in Table 3-2, with the addition of an association descriptor parameter. Responsibility for invokeId consistency is left to the user of CMISE, in a similar fashion to the ISODE ROSE. A "m-wait" procedure models indications and confirmations, following a queue model as opposed to a callback or upcall model [Clark85]. The m-wait request primitive may be instructed to simply inspect the queue i.e. return immediately, to wait for a specified period or to wait indefinitely until an indication or confirmation arrives. A relevant application could be organised in a single or multi-threaded fashion. In the case of a single-threaded execution paradigm, the incoming indications and confirmations need to be managed. The relevant mechanism is orthogonal to the MSAP API and is discussed in more detail in section 3.7.

A part of the MSAP CMIS API is shown in Table 3-3. The approach taken corresponds very closely to the CMIS standard [X710], which can serve as relevant documentation. The reader may observe the similarity between the programmatic CMIS interface of Table 3-3 and the abstract CMIS primitives of Table 3-2.

int M_Get	(int assocId, int invokeId, External* access, MIDentifier* objClass, MNane* objName, CMISScope* scope, CMISFilter* filter, CMISSync sync, int nattrs, MIDentifier attrIdList[], MSAPIndication* mi);
int M_GetRes	(int assocId, int invokeId, int linkedId, MIDentifier* objClass, MNane* objName, char* currentTime, int nattrs, CMISGetAttr attrList[], CMISErrors error, CMISErrorInfo* errorInfo, MSAPIndication* mi);
int M_Wait	(int msd, int waitPeriod, MSAPIndication* mi);

Table 3-3 (Part of) The MSAP CMIS API

3.3.2.3.3 Attribute, Action, Event and Specific Error Values

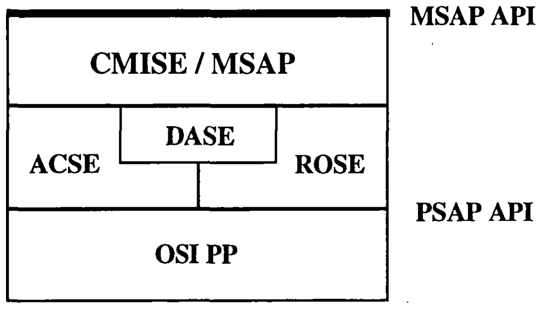
Another important decision behind a CMIS API concerns the representation of parameters with dynamic nature whose exact type is not known by the CMIP protocol. These are the attribute value, action information and reply, notification information and reply and object-level error information in the case of a processingFailure error. All these are specified as ASN.1 ANY parameters by CMIP [X711], acting essentially as place holders for information that will be defined at a higher-level, by managed object classes. Their exact definition at the CMIP level uses the ANY DEFINED BY ASN.1 construct, which associates a name to an ASN.1 type e.g. an attribute name to the corresponding type. For example, the specification of the *uxObj* class in Appendix C associates the *nUsers* attribute to the *ObservedValue* ASN.1 type defined in [X721] and the *echo* action information and reply arguments to the *GraphicString* type [X209].

When designing a CMISE API, the key question is if responsibility for encoding and decoding those values will be left to the application, as the type ANY implies, or it will be undertaken by CMISE. The second approach is obviously more user-friendly since it results in additional information hiding. It implies though that the CMISE layer should be able to determine the exact data type for an ANY value and invoke the appropriate encode / decode method. This can be accomplished through access to *meta-data* produced from the GDMO and ASN.1 specification for a particular information model. These should map attribute, action, event and specific error names to ASN.1 types and associated manipulation logic i.e. encoding and decoding procedures. An object-oriented API can unify the ASN.1 data structures and relevant manipulation logic through object classes, allowing for a natural representation of the ANY type in higher layer APIs. In fact, this is what the OSIMIS object-oriented ASN.1 API does as will be described in section 3.4.

The procedural MSAP API follows the ISODE policy, which always passes control to the API user for dealing with the ANY type. XMP [XMP] is fairly flexible, accommodating both approaches: the programmer can instruct the CMISE infrastructure to either encode/decode ANY values or leave them to be manipulated by the application. Finally, the CMIS/C++ API [Chat97] uses the separate ASN.1/C++ API which is object-oriented, supporting a natural manipulation of the ANY type.

An associated design decision has to do with the API representation of the attribute, action, notification and specific error names. These are defined at the CMIP level as ASN.1 OBJECT IDENTIFIER (OID) types, with their values defined in a GDMO specification through "REGISTER AS" clauses. For example, the *objectClass* attribute of the *top* class [X721] is registered as {joint-iso-ccitt(2) ms(9) smi(3) part2(2) attribute(7) objectClass(65)} or, more concisely, as 2.9.3.2.7.65 . It is this value that is communicated across the Q_3 interface, encoded according to the transfer syntax in use, and *not* the user-friendly string representation "objectClass". The simplest API policy is to pass the actual OID in a concrete representation e.g. a C data structure. A better API policy that results in more information hiding is to pass the user-friendly string and let the CMISE layer map it to the associated OID. In the latter case, the CMISE layer should have access to meta-data associated with a particular GDMO information model. The OSIMIS MSAP API follows the former, more "low-level" approach in order to be consistent with the ISODE API policy.

3.3.2.3.4 The OSIMIS CMISE



MSAP: Management Service Access Point
PSAP: Presentation Service Access Point

Figure 3-4 The OSIMIS CMISE Realisation

Figure 3-4 shows the layered OSIMIS CMISE realisation. When this figure is contrasted with Figure 3-2, which depicts the upper layer protocol profile for the Q_3 interface, it shows two important design decisions which have already been described above. First, the incorporation of association control primitives in the CMISE API; and second, the incorporation of location transparency features in the CMISE API through the use of DASE for accessing the OSI Directory. These decisions make the CMISE API self-contained i.e. the user does not need to either learn and or access a separate ACSE or DASE API. In OSIMIS the name MSAP describes both for the CMISE implementation, i.e. the relevant library, *and* the CMISE API.

From an engineering perspective, all the upper layer protocols, including CMISE, are realised as libraries linked with a management application. This means that each management application contains its own "instance" of the upper layer protocol stack. The lower layers are typically part of the operating system's kernel, so all the applications use a single instance of the lower layers. In a general purpose operating system such as UNIX, TCP/IP, TP4/CLNP and X.25 are part of the kernel. This means that RFC1006 in the case of TCP/IP and TP0 / TP2 in the case of X.25 run in user space, together with the upper layer stack. An evaluation of the impact of the "tightly-coupled" upper layers to the size of management applications is presented in section 3.8.

In summary, realising a CMISE API and protocol machine involves a number of important design decisions as described in this section. It is not difficult though, assuming the existence of ASN.1 tools and disregarding, at least initially, location transparency features. The author spent around 3 months for the design, implementation and testing of the bare-bone OSIMIS CMISE in late 1989. The API paradigm was procedural instead of object-oriented in order to maintain ISODE compatibility, which finally proved to be unnecessary. C. Stathopoulos of ICS implemented the location transparency features first in early 1993 and re-implemented them in 1995 to track the [X750] standard.

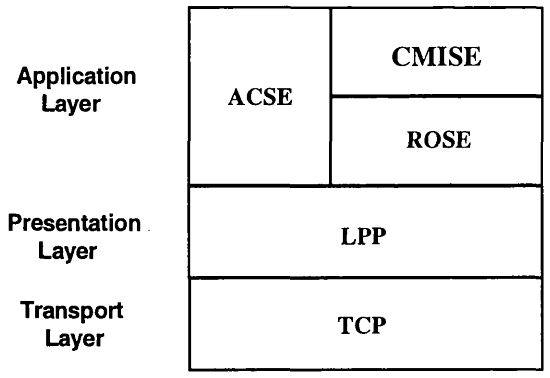
The popularity of this procedural CMISE implementation proved to be remarkable, used in many research projects and products and promoting the concept of OSI-SM as a whole. It should be finally noted that the discussion and the issues raised in this section, though targeted at CMIS/P, can be generalised for any other application service element.

3.3.2.4 Alternative Mappings for CMISE

In the previous sections we discussed the issues behind realising CMISE over a full upper layer Q_3 protocol stack [Q812]. While CMIP [X711] is specified in ASN.1 and uses the OSI ROSE [X219], it is a general management protocol that can be put over a different transport infrastructure. In fact, the whole of OSI-SM including GDMO, the SMFs and the manager-agent application framework can be adapted and used over environments other than OSI. In this section we examine the issues behind alternative mappings for CMISE.

The first key requirement for CMISE is reliable transport infrastructure. This can be provided either by the OSI TP over pure OSI lower layer protocols or by the Internet TCP/IP, in the latter case with or without the RFC 1006 packetisation protocol. The second key requirement is a presentation facility of similar expressive power to ASN.1.

As it was already mentioned, CMISE and ROSE-based ASEs do not use any of the sophisticated functionality of the session protocol. This means it should be possible to provide a lightweight mapping for an upper layer stack by using a modified version of the OSI presentation protocol operating directly over a reliable transport mechanism. This was exactly the thinking behind the mapping specified in [Rose88], which is known as the Lightweight Presentation Protocol (LPP). That particular mapping exploits the fact that BER streams are "self-delimited" because of the *tag-length-value* approach [X209], so it maps LPP directly over the Internet TCP which provides a stream-oriented reliable transport service. The LPP "hardwires" a number of parameters which are generally negotiated at association establishment and restricts the transfer syntax to be the BER.



LPP: Lightweight Presentation Protocol

Figure 3-5 The CMOT Protocol Stack

The mapping of CMISE over the LPP is shown in Figure 3-5 and is known as CMOT - CMIP Over TCP/IP [Besa89]. Since ISODE supports the LPP, OSIMIS subsequently supports the CMOT stack. This mapping is a Q_x protocol in TMN terms but it has not had much use in telecommunications environments. The key reason is that it sacrifices interoperability in comparison to the full Q_3 stack while it does not bring significant improvements to the size and performance of relevant applications, as it will be discussed in section 3.8. The LPP approach as a whole has remained mostly a paper exercise, without any real deployment.

Another approach towards lightweight mappings of OSI application layer protocols has been taken by the Lightweight Directory Access Protocol (LDAP) [LDAP]. LDAP has been recently very popular because of the commercial interest in OSI directory technology [X500] over the Internet. Its key aspects are:

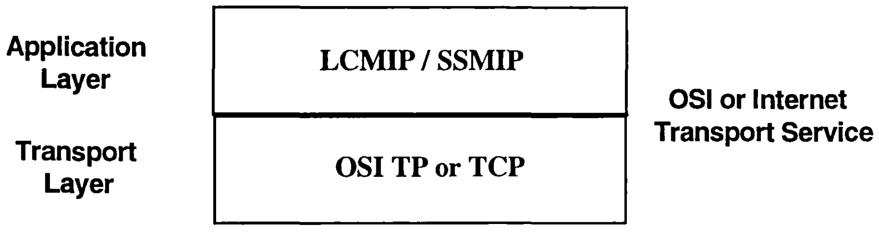
- protocol data units are carried directly over the TCP or the OSI transport service, bypassing completely both the presentation and session overhead;
- many parameters of the protocol primitives are encoded as strings e.g. distinguished names, attribute types and values, etc.; and
- the protocol data units themselves are specified in ASN.1 and encoded in BER which is used in a restricted form in order to simplify implementations.

LDAP-based applications do not need presentation facilities since they communicate attribute values in pretty-printed *string* form. LDAP is typically used for lightweight DUAs and it is relevant PDUs are converted to DAP through LDAP-DAP gateways or service relays. Many recent commercial DSAs support LDAP directly, in addition to DAP, in which case there is no need for gateways.

The author was intrigued by the principles behind LDAP and adopted similar concepts for the specification of the Lightweight CMIP (LCMIP) protocol [Pav95c]. The latter uses the same principles described above for LDAP but introduces a number of additional simplifications. While ASN.1 and BER are used to describe and encode the LCMIP PDUs, only a limited set of ASN.1 types are allowed, namely *NULL, INTEGER, OCTET STRING, SEQUENCE, SET OF* and *SEQUENCE OF*. The LCMIP PDU specification was structured in such a way as to allow maximum reusability of encoding and decoding procedures. There are no ASN.1 optional elements while numeric tags have been kept to a minimum, since they result in different encodings. The idea was to be able to implement LCMIP by hand, without the need for ASN.1 compilers which inevitably introduce inefficiencies.

Distinguished names are communicated as strings using the ISODE string convention, used also in LDAP e.g. "`logId=1@logRecordId=5`". CMIS filters use the string format described in Appendix D. Attribute, action, notification and specific error values are communicated as pretty-printed strings whose structure should be agreed. OSIMIS provides already "standard" string representations for the DMI types [X721]. New GDMO/ASN.1 specifications should always define the string representations for the ASN.1 types they introduce. For example, the MeanStdDev type defined in Appendix C could have the string representation
"`{ mean: <val> stdDev: <val> }`".

LCMIP includes a number of other optimisations that simplify the structure of the CMIP PDUs. The LCMIP structure of the GetArgument and GetResult LCMIP types is presented in Appendix E, highlighting some of the major design decisions and simplifications. The LCMIP approach was never implemented, mainly because of lack of resources but also because the use of full Q_3-capable applications proved to be less expensive than widely believed, as explained in section 3.8. It would be interesting though to be able to quantify the savings of this approach compared to the full Q_3 one, the author intends to pursue this in the future. The - very simple - LCMIP protocol stack is shown in Figure 3-6.



LCMIP: Lightweight Common Management Information Protocol
SSMIP: Simple String-based Management Information Protocol

Figure 3-6 The Lightweight and String-based CMIP Protocol Stack

While LCMIP is much simpler than CMIP and uses strings for attribute, action, notification and specific error values, its PDUs are still specified in ASN.1 while BER is used for their encoding. Taking the concept of a string-based representation further, a possibility would be to communicate the whole of the lightweight CMIP PDUs in a pretty-printed string form. The key issue in this case is the definition of this pretty-printed PDU format.

When the author initially implemented the OSIMIS CMISE and generic agent infrastructure, he also implemented a number of generic command line manager programs that provided the full functionality of the CMIS primitives, namely *mibdump* (or *mget*), *mset*, *maction*, *mcreate*, *mdelete* and *evsink*. Their syntax, which follows the UNIX convention for command line arguments, is described in [Pav93b] and realises essentially a string form for the CMIS request primitives. Since these programs parse their input, the relevant logic can be reused as part of a CMIP protocol machine. A full string-based CMIP requires also the specification of the string form for the reply and error PDUs, which can be based on the same principles.

Table 3-4 presents the PDU structure of a string-based lightweight CMIP protocol which is largely based on the syntax of the OSIMIS generic command-line manager programs [Pav93b]. The author named this Simple String-based Management Information Protocol (SSMIP). Figure 3-6 shows the protocol mapping which exactly the same as that of LCMIP. The SSMIP should be more lightweight than LCMIP while its big advantage is that it can be implemented without the need for ASN.1 compiler support. Its PDUs can be encapsulated in protocols such as HTTP so that it can drive WWW displays. The SSMIP was never implemented in OSIMIS but a variation of it was implemented in a commercial product which is based on OSIMIS.

mcre invId [-A access] -c class [-n name | -s superiorName] [-r referenceName] [-a attrName=value ...]
mcgt invId
mget invId [-A access] [-c class] [-n name] [[-s scope] sync] [-f filter] [-a attrName ...]
mset invId [-A access] [-c class] [-n name] [[-s scope] sync] [-f filter] [-[w|d|a|r] attrName[=value] ...]
mact invId [-A access] [-c class] [-n name] [[-s scope] sync] [-f filter] -a attrName[=value]
mdel invId [-A access] [-c class] [-n name] [[-s scope] sync] [-f filter]
mevr invId [-c class] [-n name] [-t time] -a eventName[=value]

mres <op>[2] invId [-l linkId] [-c class] [-n name] [-t time] [-a [-e error] [name[=value] [-m modify]] ...]
merr invId error [<errInfo>[3]]

Table 3-4 String-based CMIP PDUs

In summary, comparing the LCMIP and SSMIP approaches, it is worth going all the way and adopting the SSMIP approach as opposed to the LCMIP one since it uses a simpler, fully string-based approach. Such protocols are useful for driving TMN WS applications [Pav96d] which typically manipulate management information in string form. They may not be particularly good for applications that examine management information and perform numerical calculations. In those cases, a significant amount of processing time will be spent in converting management information from numeric to string form and vice-versa. It should be finally mentioned that when these protocols drive TMN WS applications, they may be thought as "proprietary" F protocols. One of the reasons we have proposed not to standardise the TMN F interface in Chapter 2 is because there exist many different styles of string-based CMIP protocols over various different transport mappings. WS applications that use SSMIP should communicate with the rest of the TMN through service relays that convert SSMIP to the Q_3 protocol stack.

While in this section we have considered alternative mappings for CMISE that use simple string-based representations of attribute, action, event and specific error values, the applicability of those protocols is typically restricted to workstation applications as explained above. In Chapter 4 we examine a more general mapping to OMG CORBA and distributed object technologies.

[2] The operation type is the name of the operation i.e. mcre, mcgt, mget, mset, mact, mdel, mevr.

[3] The error information has structure that is specific to the particular error code.

3.3.2.5 Summary

In this section we described in detail the issues behind realising the protocol part of the TMN Q_3 interface and examined possible policies for the relevant API. Since CMIS/P requires only a reliable transport service and a presentation facility, it can be mapped onto alternative transport infrastructures as discussed in section 3.3.2.4.

In summary, CMIP is a modestly complex protocol that can be relatively easily realised. The necessary infrastructure should be an OSI development environment that includes a procedural ASN.1 compiler. An aspect that makes the protocol more complex than necessary is the use of object identifiers instead of user-friendly string names. This is something pertinent to all the OSI applications and introduces unnecessary complexity. Proponents of the solution claim that it provides guaranteed uniqueness of names, which is true. On the other hand, such uniqueness could be policed by a central authority that would endorse new GDMO specifications.

While CMIS services are relatively easy to provide, the real difficulty lies in providing a development environment that hides CMIS/P and provides an object-oriented distributed platform that supports the rapid development of TMN applications by developers with little or no knowledge of network programming. The relevant issues are examined in sections 3.5 and 3.6, after we examine issues on object-oriented ASN.1 manipulation in the next section.

3.4 Issues in Object-Oriented ASN.1 Manipulation

A important aspect of OSI upper layer is the manipulation of ASN.1 data structures. Typically, ASN.1 compilers map those abstract data structures to concrete programming language representations, as already discussed in section 3.3.2.1. This mapping can be procedural, with separate data structures and syntax manipulation procedures, or object-oriented, with classes mapped to ASN.1 types and relevant syntax manipulation methods. An important issue in the latter case is the polymorphic design of the relevant API. OSIMIS is based on ISODE which supports a procedural ASN.1 manipulation style through the *pepsy* ASN.1 compiler, in a similar fashion to most ASN.1 infrastructures of the late eighties and early nineties [Neuf90]. As such, it has been necessary to define object-oriented ASN.1 abstractions in OSIMIS and to provide an object-oriented ASN.1 compiler with C++ mappings. The issues behind high-level object-oriented ASN.1 manipulation are discussed in this section.

There is very little work in the literature on issues related to flexible high-level ASN.1 APIs, as opposed, say, to work on ASN.1 performance measurements and comparisons. One well-known approach to ASN.1 manipulation is X/Open's XOM API [XOM] that has been described in the previous section. Its key drawback is that it is object-based as opposed to object-oriented. One of the main reasons behind the fact that the XOM/XMP API has been rather unpopular has to do mostly with the XOM rather than the XMP part.

Another more recent and much more promising approach is the NMF ASN.1/C++ API, which is part of the overall TMN/C++ series of APIs [Chat97]. This maps ASN.1 types to C++ classes that derive ultimately from the abstract class `AbstractData`; the latter heads the C++ class hierarchy for ASN.1 types. This class provides functionality inherent in ASN.1 data types, such as encode, decode, print, compare, discover its type information etc. For each ASN.1 built-in type, a C++ subclass of AbstractData provides a type-specific representation, e.g. Boolean, Integer, Sequence, etc. Other ASN.1 types map to classes derived from those. The whole approach is in fact extremely similar to the one designed by the author and described in this section. It should be noted that the OSIMIS approach was passed as input to the NMF TMN/C++ team.

Before we describe our approach, we need to clarify further some issues behind the representation and use of the ASN.1 ANY type in upper layer infrastructures. The ANY type is typically used to pass "unknown" types between layered ASEs. For example, the CMISE m-get PDU is of

ASN.1 type GetArgument [X711] but is passed to ROSE [X219] as an ANY type. This is because ROSE is unaware of higher-level ASEs, so it specifies the arguments and results of its operations as of type ANY. The extensive use of the ANY type in upper layer infrastructures implies that a relevant API representation is necessary. One may suggest that this should be a byte stream, encoded according to the transfer syntax in use. The problem with this approach is that it increases the complexity for the relevant applications since they have to deal explicitly with encoding and decoding. It also violates the layering principle since the application layer becomes explicitly aware of the transfer syntax, which is normally a function of the presentation layer. In addition, the logic produced by the ASN.1 compiler becomes dependent on the particular transfer syntax. This reduces flexibility, in the sense that a different transfer syntax can be supported only after recompiling the application software.

A generic procedural approach for representing the ASN.1 ANY type was first pioneered in ISODE [ISODE][Rose90]. According to this, a special data structure can represent any ASN.1 type in a transfer syntax independent fashion. This can be generally termed an "intermediate ASN.1 representation" and in ISODE it is specifically called a Presentation Element (PE). ASN.1 compilers produce logic that converts a concrete representation to an intermediate one and vice-versa (*encode* and *decode* a type). Such generic structures can be passed through the upper layer APIs and can be *serialised* (and *de-serialised*) in the presentation layer, according to the relevant presentation context. This is exactly the policy followed in the ISODE ASEs and in many other non object-oriented OSI infrastructures.

The key ingredient of high-level TMN APIs is the object-oriented manipulation of ASN.1. The author realised this early in the initial design of OSIMIS (around 1990). This led to the design of the generic *Attr* class, whose polymorphic interface defines the rules for generic object-oriented ASN.1 manipulation in OSIMIS [Pav93a][Pav93b]. Every ASN.1 type is modelled by a class that derives either directly from Attr, or indirectly through another generic class such as Enumerated, Integer, String, List, etc. In addition, the *AnyType* class models specific types in a generic fashion and is typically used by generic manager applications. Finally, the generic *AVA* class (Attribute Value Assertion) was added later to model the *ANY DEFINED BY* ASN.1 construct, which associates attribute, action, event, and specific error names to ASN.1 types. An O-O ASN.1 compiler wraps up the output of the ISODE pepsy compiler and produces C++ classes for specific ASN.1 types.

```
class Attr
{
protected:
    virtual PE       _encode ();
    virtual void*    _decode (PE);
    virtual char*    _print ();
    virtual void*    _parse (char*);
    virtual void     _free ();
    virtual void*    _copy ();
    virtual int      _compare (void*, void*);
    virtual void**   _getElem (void*);
    virtual void**   _getNext (void*);
    // . . .
    Attr ();         // abstract class

public:
    virtual char*    getSyntax ();
    Bool             isMultiValued ();

    PE               encode ();
    char*            print ();
    void             ffree ();
    void*            copy ();
    // . . .
    void*            getval ();
    void             setval (void*);
    int              setstr (char*)

    virtual Bool     filter (int, void*);

    void             clear ();
    virtual          ~Attr ();
    // . . .
};
```

Code 3-1 The Generic Attr Class that Models an ASN.1 Type

We will start discussing the aspects of object-oriented ASN.1 manipulation by examining the features of the Attr class, which realises the fundamental aspects of the ASN.1 API. Attr is an abstract class which is never instantiated but serves as the root of the relevant C++ class hierarchy, in a similar fashion to the AbstractData class in the ASN.1/C++ API [Chat97]. The O-O ASN.1 compiler produces automatically derived classes that the model specific ASN.1 types e.g. Integer, OperationalState, etc.. The Attr class encapsulates the relevant data type while derived classes redefine the associated manipulation functions. It comprises the following polymorphic manipulation methods:

- _encode and _decode, which convert to and from the intermediate representation;
- _print and _parse, which convert to and from a pretty-printed string;
- _free, which releases memory and _copy, which makes a copy;
- _compare, which compares two instances of the encapsulated data type; and
- _getNext and _getElem which can be used to walk through a multi-valued type (ASN.1 SET OF or SEQUENCE OF) and access the contained elements.

All these methods can be produced automatically through an ASN.1 compiler. The produced print and parse functions may use a "not-so-pretty" string representation while the _compare method may not be able to exploit "buried in" semantics of a particular type , e.g. order comparisons for a "time" type. The user might want to overwrite those methods through manually supplied ones or even add new methods. Such a facility needs to be supported by the relevant O-O ASN.1 compiler. It should be noted that CMIS filtering is automatically supported through the filter method. Finally, it is possible to use string representations to construct and manipulate a type e.g. Integer("5"), AdminidtrativeState("locked"). It is also possible to use intermediate representations, e.g. ISODE PEs, which is important for constructing relevant objects when ascending the protocol stack, i.e. in indications and confirmations.

A number of additional generic classes model generic properties of a "family" of types, such as enumerated, null-terminated string, list, etc. An example inheritance hierarchy is depicted in Figure 3-7 in OMT notation. The *count*, *gauge*, *threshold* and *tide-mark* classes model the relevant types defined in [X721]. It should be noted that those types have well-defined behaviour which has to be hand-written. They can be implemented once though and be subsequently re-used.

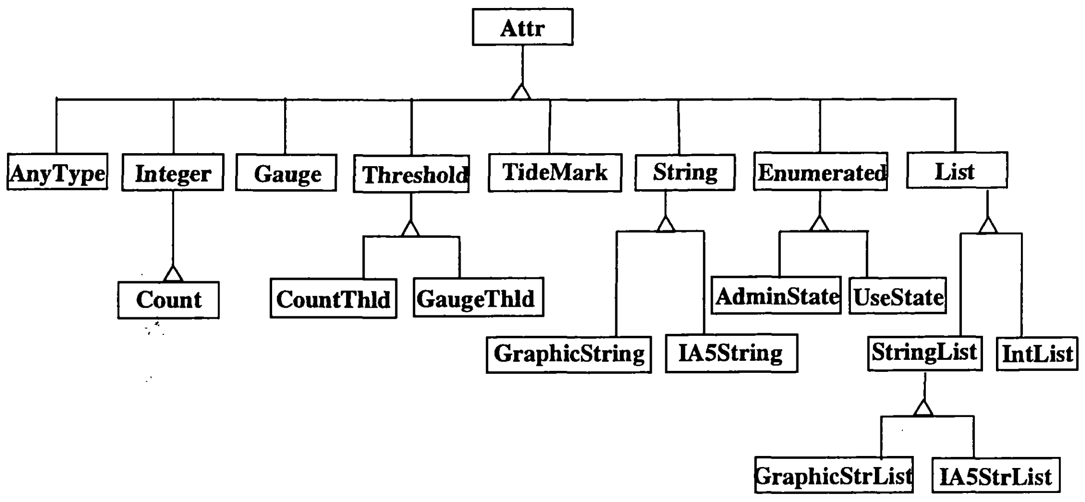


Figure 3-7 Example ASN.1 Class Hierarchy

In an object-oriented CMISE protocol stack, values of the "unknown" ANY type i.e. attribute, action, event and object-specific error values, can be passed through the API as Attr parameters. These can descend the stack in the case of requests and responses, carrying with them behaviour to produce the required intermediate representation at some point. In the case of indications and confirmations, the CMISE layer needs to know which ASN.1 type corresponds to a particular name so that it can construct the corresponding object e.g. *Count* for the *bytesSent* attribute. This can only be achieved if the CMISE layer has access to *meta-data* for a particular GDMO/ASN.1 object model. The exact information required is the mapping of the relevant name to the corresponding ASN.1 type. This information can be produced by GDMO compilers and stored in some form of database that implements an information model *repository*. We will discuss issues on such a repository in section 3.5.6.

In management environments, there is a particular class of manager applications which are information model independent in the sense that they do not depend on the semantics of the particular model they access. An example of such an application is a MIB browser [Pav92a] which allows a human user to browse through the MIT across a management interface and set attribute values, perform actions and create/delete managed objects. Such an application is written once, in a generic fashion, and needs simply to be "updated" (i.e. linked with in software terms) with the particular ASN.1 syntaxes for a new GDMO information model [Pav92a]. Such an application cannot use the specific C++ classes that model particular syntaxes, e.g. Integer, GraphicStringList etc., since it is supposed to deal with any future types introduced by new GDMO models. This necessitates the introduction of a generic type, the *AnyType*, which is a generic specialisation of the Attr class (see Figure 3-7). This type implements its functionality by having access to meta-data, which include in this case the encoding, decoding, printing and parsing procedures produced by the ASN.1 compiler or supplied by the human user. This type is of paramount importance for generic applications and was conceived early in OSIMIS, together with the Attr class. It should be noted that the NMF NMF/C++ API [Chat97] includes such a facility.

Finally, the mapping of arbitrary ASN.1 types to names is modelled in ASN.1 via the ANY DEFINED BY construct. This is a powerful feature but can also be easily misused. It can be thought as the ASN.1 equivalent to the "void pointer" in C and C++. For example, the programmer may map the wrong type to a particular name which will be transmitted correctly across the network but will most probably result in an obscure error produced by the other end (or in a core dump in the case of not-so-bullet-proof software!). In order to harness the relevant power, the author designed the Attribute Value Assertion (AVA) class, whose salient features are depicted in the Code 3-2 caption.

```
class AVA
{
    // . . .
public:
    static Bool       createError ();

    char*             getName ();
    Attr*             getValue ();
    CMISErrors        getError ();
    CMISModifyOp      getModifyOp ();

    char*             print ();
    void              clear ();
    // . . .

    AVA (char*, void*, CMISModifyOp = noModifyOp);
    AVA (char*, char*, CMISModifyOp = noModifyOp);
    AVA (char*, Attr*, CMISModifyOp = noModifyOp);
    AVA (OID, PE, CMISErrors, CMISModifyOp);
    // . . .
    ~AVA ();
};
```

Code 3-2 The AVA Class

The AVA class encapsulates the relevant syntax object together with the corresponding name. It also encapsulates a modify operator, used in CMIS m-set requests, and an error value, used in CMIS results to denote some error condition e.g. attribute not set because of access, invalid value or other problem. When constructing a AVA instance, the consistency of the name and type are checked through access to the information model repository, so the programmer is protected in the case of an error. Both the Attr and AVA classes are used extensively in the OSIMIS high-level APIs.

The Code 3-3 caption shows some of the power and expressiveness of the object-oriented ASN.1 API through brief examples. The latter show the manipulation of the base ASN.1 type INTEGER through C++ classes. Initially, an *Integer* instance is constructed, first through the encapsulated data structure, i.e. the C++ *int* built-in type, and then through an equivalent string value. The encapsulated data structure can be accessed either through the generic Attr::getval() method or through the type-specific user-supplied Integer::getint() method. The next example shows the construction of a generic AnyType instance that realises an integer with the same value as before. The type can be specified either explicitly, e.g. “Integer”, or implicitly through the associated name, e.g. “bytesSent”. Note that the value may be printed without any knowledge of the encapsulated data structure. This form of manipulation is typical in generic manager applications. Finally an AVA instance is constructed and printed, first by using explicit type knowledge and then generically. Programming with explicit types has the advantage of additional type-specific methods which may be manually supplied, e.g. Integer::getint(). It is also more performant since there is no need to access meta-data at construction time, as it is the case with the AnyType class.

```
// Integer construction with the encapsulated data structure

int* val = new int;
*val = 10;
Integer* i = new Integer(val);

// Integer construction with string

Integer* j = new Integer("10");
cout << "j is " << j->getint() << endl; // getint() is user-added

// generic AnyType construction based on type knowledge (Integer)

AnyType* k = new AnyType("Integer", "10");
char* cp;
cout << "k is " << cp = k->print(); // generic print
delete cp;

// generic AnyType construction based on name knowledge (bytesSent)

AnyType* l = new AnyType("bytesSent", "10");
cout << "l is " << *(int*) l->getval() << endl;

// AVA construction based on name (bytesSent) and value (Integer)

AVA*     m = new AVA("bytesSent", i);
cout << "m name is " << m->getName();
cout << " and value is " << *(int*) m->getValue->getval() << endl;

cout << "m is " << cp = m->print() << endl; // prints name: value
delete cp;
```

Code 3-3 Example Use of the O-O ASN.1 API

We will finalise this section with a brief discussion of the realisation issues. Ideally, one would like to implement an ASN.1 compiler that produces directly the relevant C++ classes for the various types. Since users might also want to overwrite some of the produced methods, such a facility needs also to be supported. Unfortunately, most OSI upper-layer environments come with their own, typically procedural, ASN.1 compilers which follow their own conventions. One may try to encapsulate their output, which is what the author has done with the ISODE pepsy compiler, but there are limits as to how far one can go with this approach. For example, the C structures produced by a procedural compiler are still visible in the C++ API presented above and have to be manipulated by programmers. Removing all the dependencies on a procedural ASN.1 API means essentially implementing a new native object-oriented ASN.1 compiler and this can be *a lot* of work (ASN.1 is a pretty complex language). The "wrap-up" approach though is acceptable and can go a long way towards supporting object-oriented APIs for ASN.1 manipulation.

The OSIMIS O-O ASN.1 compiler wraps-up the output of the ISODE pepsy compiler. It performs first some rudimentary parsing of the ASN.1 input file, invokes the pepsy compiler and parses partially the output of by the latter. It knows the ASN.1-to-C data structure conventions used by pepsy and uses this and the parsing information to build a symbol table and to drive the generation of C++ classes that correspond to the ASN.1 types. User's code that overwrites produced methods can also be incorporated. The process of producing C++ classes for ASN.1 types is shown in Figure 3-8. This wrap-up compiler is a modestly complex program, written using the GNU version of the UNIX awk tool [Kern84]. The latter provides a very flexible "interpreted C"-like facility. The author spent about a month for the relevant implementation. On the other hand, it took almost a year to realise the possibility for such a solution and work out the relevant details. The O-O ASN.1 compiler was implemented in the course of 1994.

The heart of the ASN.1 API is the Attr class whose polymorphic interface took quite some time to finalise. A first version was designed and implemented in 1990 with the early version of OSIMIS but it took a number of iterations to get it right. It is difficult to quantify such a task in terms of complexity and time, since it involves very delicate aspects of polymorphic object-oriented design. The same is true for all the high-level object-oriented OSIMIS APIs. Finally, the AVA class was conceived and added in 1993, while designing the RMIB high-level manager API; the latter will be described in section 3.5.

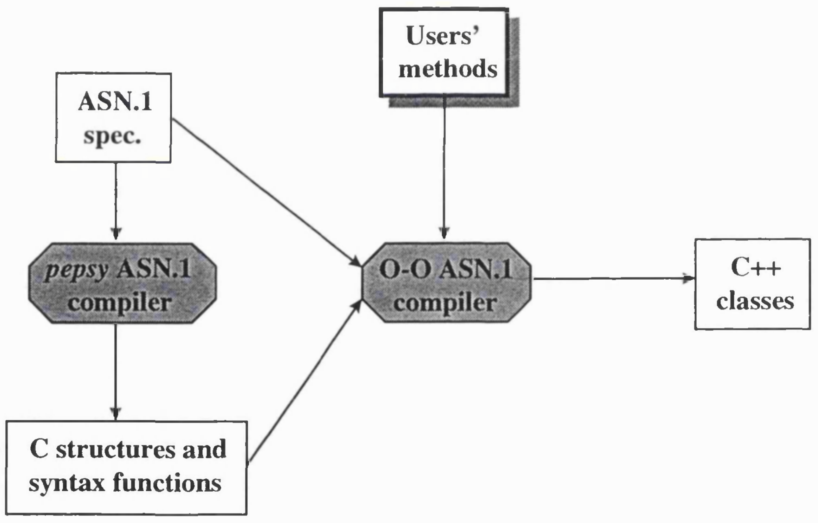


Figure 3-8 C++ Class Generation for ASN.1 Types

In summary, object-oriented ASN.1 manipulation is key for user-friendly upper layer APIs. The author recognised early the need for such a facility and designed and implemented first the relevant classes and later the O-O ASN.1 compiler. For a number of years, the only available compilers and APIs were procedural while there is very little research work discussing the issues behind object-oriented ASN.1 APIs. The need for such an API was recognised in the mid-90's by the NMF TMN/C++ team, with the author's approach being an input to that work. Their ASN.1/C++ solution [Chat97] bears a lot of similarities to the approach presented here, which was conceived, designed, realised and made publicly available much earlier.

3.5 Issues in Realising Object-Oriented Manager Infrastructures

3.5.1 Introduction

In section 3.3.2.3 we concentrated on issues related to the realisation of CMISE and discussed policies for the relevant API. The author chose to design a relatively "low-level", procedural CMISE API in accordance to the ISODE conventions, the latter being the supporting OSI upper layer development environment; this API is known as MSAP. Such an API can be cumbersome and difficult to use while it typically results in increased code size for management applications. As already explained, the author adopted such an approach for non-technical reasons and had already in mind higher-level, object-oriented APIs while designing the MSAP one. In fact, developers of OSIMIS-based TMN applications that use those object-oriented APIs are not aware of the MSAP API at all.

Because of the manager-agent duality, there exist two types of higher-level APIs: APIs in manager roles, which should provide access to managed objects in a user-friendly, high-level fashion, hiding CMIS/P access details but not sacrificing its expressive power. And APIs in agent roles, providing an environment for the realisation of managed objects which hides the underlying CMIS/P access aspects, allowing implementers to concentrate in the realisation of the associated behaviour. It should be noted that this separation is not specific to the manager-agent model and the OSI-SM/TMN information architecture but it is an inherent aspect of any distributed object-oriented environment that follows the client-server model. For example, the OMG CORBA [CORBA] mapping of the relevant abstract language to concrete O-O programming languages has also two distinct client and server facets.

In this section, we concentrate in high-level object-oriented infrastructures and associated APIs for applications in *manager* roles. There can be two types of such APIs: those that provide an object-oriented abstraction of an application in agent role; and those that provide object-oriented abstractions of its individual managed objects. In both cases, by object-oriented abstractions we mean objects in local address space which model either an agent application or an individual managed object, in a "proxy" fashion. One may remark that CMISE APIs model to some extent an agent application. This is true, but none of the existing CMISE APIs, i.e. the OSIMIS MSAP [Pav93b], the X/Open XMP [XMP] and the NMF CMIS/C++ [Chat97], provide the level of abstraction and functionality we will propose here.

3.5.2 Related Work

There is less research work on higher-level manager APIs, as opposed to research work in designing and developing agent infrastructures and APIs, which will be considered in section 3.6. In particular, there is very little research work on abstractions modelling remote agents, similar in scope and functionality to the proposal of the author detailed in section 3.5.3. It seems that relevant research work either concentrates in modelling raw CMIS functionality, e.g. the X/Open XMP [XMP], the NMF CMIS/C++ [Chat97], or addresses directly abstractions at the managed object level, e.g. the IBM Object-Oriented Interface (OOI) [Holb95], the NMF GDMO/C++ [Chat97] and even CORBA [CORBA][BenN94].

The IBM Object-Oriented Interface (OOI) [Holb95] presents an approach for modelling remote managed objects in a manager application through Proxy Managed Objects (PMOs) associated to a Proxy Agent object that models the remote agent. The latter provides an object-oriented view of CMIS, realised over XOM/XMP. It encapsulates the XMP "session" and provides access to XMP operations through its methods, the key advantage being the use of an O-O ASN.1 API which is also part of the OOI framework. The proxy agent provides methods for accessing multiple objects through scoping, filtering and may create a number of PMOs as a result of those operations. It also allows manager objects to request event reports and supports a relevant "event-queue" facility. In summary, the Proxy Agent provides facilities similar to some extent to those of the Remote MIB (RMIB) agent proposed by the author. It should be noted that the author's work [Pav94b] predated the relevant work by IBM and has been taken into account in the latter.

The PMOs model remote managed objects and the emphasis is on strong typing that results in a friendlier interface for novice application developers, providing compile-time checking. Specific PMO classes, produced through a GDMO compiler, include methods for accessing attributes and invoking actions. Both the attribute access methods (get, set) and the action methods take as parameters the precise C++ classes that model ASN.1 types according to the GDMO/ASN.1 specification. In addition, a weakly-typed generic PMO class is also provided which can be used in generic applications such as MIB browsers. This uses a generic ASN.1 type, similar to the AnyType presented in section 3.4. When weak-typing is used, run-time checking is supported through access to meta-data. IBM's approach was the main input to the NMF TMN/C++ API team for the object oriented manager API. As such, it has influenced strongly the latter which is described below.

The NMF GDMO/C++ API [Chat97] is a complete approach towards such an API. It is to an extent a superset of the IBM OOI one, incorporating also aspects from other proposals, including the OSIMIS high-level manager APIs. A remote agent is modelled through an Agent Handle (AH) class, through which multiple managed objects can be addressed through scoping and filtering. A MO is modelled through a Managed Object Handle (MOH) class, which is specific to the particular GDMO class and is produced automatically by the GDMO compiler. A Managed Object Handle Factory (MOHF) contains a set of "machine objects", one for each of the GDMO classes the manager knows about. New MOHs can be created through the factory for a particular Agent Handle or as a result of linked replies from scoped operations. The whole framework is strongly-typed, providing compile-time error checking,. There is also a weakly-typed version which can be used in generic manager applications. Finally, the user may introduce specific MOH-derived classes which implement particular policies with respect to the relevant remote MO e.g. attribute value caching, periodic updating, etc.

Although CORBA will be introduced formally in Chapter 4 and despite the fact that its underlying protocol is not CMIP/Q_3, it is worth examining briefly its client or "manager" API. In CORBA [CORBA][BenN94], remote objects are accessed through object references which point to a local proxy object. The latter is similar to the NMF GDMO/C++ MOH, providing strongly-typed access to either general or to attribute access methods (get, set). Here, there is no notion of agents since these do not exist in the CORBA framework. A difference between CORBA proxy objects and MOHs is that every access to a proxy object results in an operation to the relevant master object, while this is not the case with a MOH. In addition, in CORBA it is not possible to implement access policies "inside" a proxy object through inheritance. Finally, weakly-typed access is possible through the Dynamic Invocation Interface (DII).

3.5.3 The Remote MIB Manager Infrastructure

The need for a higher-level manager API was identified early while designing and developing OSIMIS. Having developed first the Generic Managed System (GMS) agent support infrastructure which will be described in section 3.6, it became clear that a similar environment was required for manager applications. The initial approach for developing those was to use the native MSAP CMISE API. This necessitated low-level CMIS primitive manipulation and resulted in a lot of complex code for management applications.

The first relatively sophisticated TMN system that was developed using OSIMIS was the RACE NEMESYS service management system during 1991. This comprised a combined Q-Adapter / Mediation device for an ATM simulator, an Element Manager OS and a Service Manager OS, organised in a hierarchical fashion [Pav91b][Pav92b]. The two OSs were going to be designed and developed by people who had little or no experience of network programming. It was necessary to provide them with high-level abstractions, so that the author investigated the relevant issues and designed the framework for two infrastructures:

- the Remote MIB (RMIB) agent level infrastructure; and
- the Shadow MIB (SMIB) managed object level infrastructure.

J. Cowan of UCL implemented the first embryonic version of the RMIB infrastructure in 1991. This version served as a proof of concept and was successfully used in the NEMESYS project, allowing relatively inexperienced people to develop (the manager parts of) TMN OSs. It was though skeletal and incomplete and, as such, it was not released with OSIMIS-3.0 [Pav93b]. A more complete approach to the design and implementation of the RMIB infrastructure took place during 1993 in the RACE ICM project. The author together with T. Tin of UCL revised the model and redesigned the relevant API. T. Tin subsequently implemented the RMIB infrastructure which was publicly released with OSIMIS-4.0 [Pav95b].

3.5.3.1 Design Issues and Objectives

One of the key objectives while designing such an infrastructure was to hide the intricacies of the underlying communication infrastructure as much as possible, without sacrificing any of the available expressive power. This implies a genuine object-oriented abstraction of the OSI management access service which would be programmer-friendly and easy to use, making possible the development of TMN applications by users with very little or no knowledge of network programming. We examine in detail below the sub-objectives accruing from this main objective.

As already explained, most CMISE APIs leave the implementer to deal with the low-level mechanics of management information access. Managed object class, attribute, action, notification and specific error names are typically passed across the API as Object Identifiers (OIDs), leaving to the user the responsibility to convert from and to user-friendly strings. Managed object names (i.e. distinguished names) are passed across as list-like data structures or objects, while the user typically deals with names as strings. The CMIS filter parameter is passed again as a complex data structure while the user would like a string-based, symbolic manipulation paradigm. In the case of non object-oriented CMISE APIs such as the OSIMIS MSAP, attribute, action, event and specific error values are passed across encoded in an intermediate representation. It is the responsibility of the API user to encode and decode native data structures to this representation.

In addition to the relatively low-level parameters to primitives, the CMIS service is by nature asynchronous and this is the way it is typically modelled in relevant APIs e.g. the OSIMIS MSAP and the NMF CMIS/C++. In this case, the application is responsible for assembling linked replies and identifying the empty PDU that denotes the termination of a particular series. While an asynchronous remote execution model is necessary in single-threaded execution environments, it requires state information to be kept by the application. A synchronous execution paradigm with RPC-like semantics is more natural to programmers, so a synchronous mode of CMIS operations should be supported. In this case though, a multi-threaded execution paradigm is necessary for increased performance. A discussion on the issues of synchronous vs. asynchronous execution paradigms for distributed management applications will be presented in section 3.7.

A key aspect of OSI-SM / TMN is their event-driven management approach. Particular events may be requested at a fine level of granularity by setting relevant filters in Event Forwarding Discriminator (EFD) objects as explained in Chapter 2. The manager application needs to explicitly manipulate EFDs while some "dispatching" mechanism is necessary to deliver an arriving event report to the right manager objects inside that application. This functionality could be undertaken by the supporting infrastructure. Finally, association management could be supported transparently by the infrastructure but the facility to explicitly initiate, terminate and abort associations should be also available to applications.

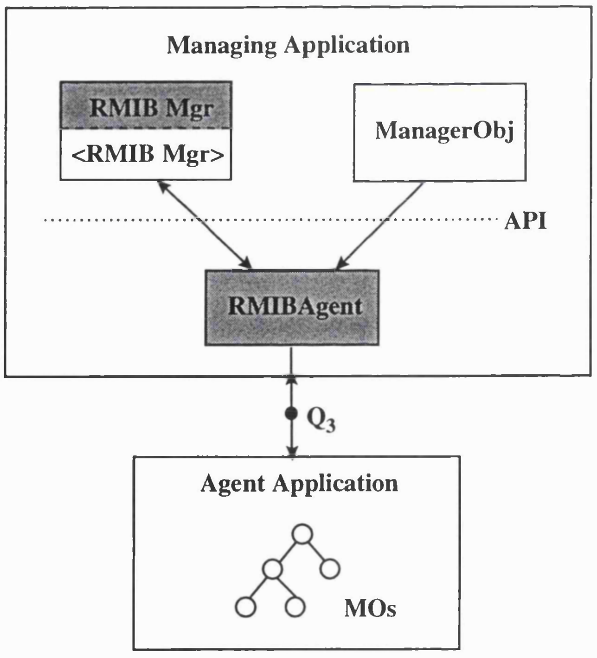
In summary, a high-level manager API has the following requirements as presented above:

- attribute, action, event and specific error names should be manipulated through user-friendly strings instead of object identifiers;
- attribute, action, event and specific error values should be manipulated through objects that correspond to the relevant ASN.1 type, rendering unnecessary any explicit encoding and decoding manipulation;
- object names and CMIS filters should be manipulated in a symbolic, string-based fashion;
- both asynchronous and synchronous remote operation paradigms should be supported;
- linked replies should be assembled in the synchronous mode; in the asynchronous mode, both an assembly of the whole series and a "n-by-n" callback facility should be supported;

- a high-level facility for the requesting and delivery of event reports should be supported, hiding the explicit manipulation of EFDs;
- association management should be transparent but also explicit control of associations should also be supported; and
- location transparency should be supported, in a similar fashion to CMISE APIs.

3.5.3.2 The RMIB Model

RMIB [Pav94b] is an object-oriented model that encapsulates the agent and associated MIB of a remote management interface. This is done through a class called *RMIBAgent* since it acts as an agent within the manager application for the remote system. It should be noted that the term "remote" implies logical as opposed to physical remoteness. For example, the co-operating manager and agent applications can be co-located at the same network node, interworking either through a full Q_3 interface or through a more lightweight interprocess communication i.e. a q_3 reference point in TMN terms.



<RMIBMgr> - specific class derived from RMIBMgr

Figure 3-9 Remote MIB Model and Interactions

Typically there exists one RMIBAgent instance within a manager application that corresponds to a particular remote agent, modelling a "binding" to that system. It is of course possible to instantiate more than one RMIBAgent for the same remote agent. The RMIBAgent class provides methods that correspond to the manager role CMIS primitives i.e. m-get, m-set, m-action, m-create, m-delete and m-cancelGet, methods to request, terminate, suspend, resume the forwarding of event reports and methods for association management. The forwarding of event reports is inherently asynchronous while remote operations on managed objects are supported in both a synchronous and an asynchronous fashion. An abstract *RMIBManager* class provides the interface for the asynchronous "callbacks" or "upcalls" [Clark85], which should be specialised in derived classes. The relationship between the RMIBManager and RMIBAgent instances is many-to-many.

The RMIB model and relevant interactions is shown in Figure 3-9, in the form of co-operating object instances. In case a manager object does not need callbacks because it is not interested in event reports and uses only the synchronous operation facility, then it does not need to be of type RMIBManager. The shaded parts of the diagram indicate generic re-usable classes. Note that the RMIBAgent class encapsulates the CMISE functionality, which in the case of OSIMIS is provided by the MSAP procedural realisation. An OMT relationship of the relevant classes is also shown in Figure 3-11 in the next section, depicting both the RMIB and SMIB models.

Aspects of the RMIB model have been adopted in the IBM OOI [Holb95] and the NMF GDMO/C++ [Chat97] approaches. A difference is that these support both callback and event-queue facilities for asynchronous operations, while the RMIB supports only a callback model[4]. Another more important difference is that those models combine a RMIB and SMIB approach in *one* framework. For example, as a result of scoped operations on the equivalent remote agent object, a set of new "shadow" objects are created which can be then accessed then locally in order to retrieve the results. The OSIMIS RMIB approach has been deliberately kept separate from the more sophisticated SMIB approach, the latter being a clear superset.

The requirements presented above are all met with the above model and associated design. Distinguished names follow the ISODE string notation e.g. "`logId=1@logRecordId=5`". CMIS filters follow a special string notation [Pav93b] which the author devised together with S. Bhatti of UCL. The latter implemented the relevant conversion logic to and from the native complex data structures. An example filter is:

"`( (objectClass=log) & (!(logId=1))& (eventType=stateChange) )`".

A complete description of the string "language" for CMIS filters can be found in Appendix D.

[4] In OMG CORBA [CORBA], the callback and event-queue models are called the "push" and "pull" model respectively.

Attribute, action, event and specific error values are passed through the Attr and AVA classes of the O-O ASN.1 API. Both synchronous and asynchronous operations are supported. In the case of scoped asynchronous operations, the relevant RMIBManager may request the assembly of linked replies which will be passed to it in a single callback. It may also request separate callbacks for every linked reply or even for every group consisting of *n* linked replies as they arrive. In the asynchronous case, the specific RMIBManager is able to exercise the m-cancelGet facility.

Associations can be either explicitly manipulated or left to the infrastructure i.e. the RMIBAgent. In the latter case, the application may manipulate a time-out for "caching" an association. If no traffic to the remote real agent is encountered during that period, the association is terminated and re-established when the next operation to that agent is requested. Location transparency is supported through the OSI Directory [X750] in a similar fashion to CMISE as described in the section 3.3.2.3.

The RMIB model supports programming with local rather than global distinguished names. The RMIBAgent is identified through the remote agent's name e.g. the full directory name `c=UK@o=UCL@ou=CS@cn=NM-OS` or simply `NM-OS` when in the local domain. Subsequent operations on managed objects through the RMIBAgent should use local names e.g. `logId=1@logRecord=5`. This is in accordance with the TMN model in which managed object clusters are manipulated together, as explained in Chapter 2. A TMN OS is typically configured with the names of other OSs it needs to access instead of global names of individual managed objects. The RMIB model supports naturally this mode of operation.

The RMIBAgent supports also a high-level event reporting API. Event reports may be requested through a string filter, they can be subsequently suspended, resumed and finally terminated. The RMIBAgent needs to create and manipulate one or more EFDs with the event filtering requirements of the RMIBManager instances. with a. When an event report is received, the RMIBAgent needs to identify the relevant RMIBManager and "push" the event report to it through an upcall. The necessary "demultiplexing" has revealed an important limitation in CMIS: it is not possible to distinguish which EFD an event report originates from since the CMIS m-eventReport primitive does *not* contain the name of the relevant EFD that triggered it. This makes impossible to demultiplex the event reports arriving at a manager application.

The solution adopted in the RMIB design overcomes this limitation in the following fashion. A separate EFD is created for each RMIBManager, with a filter which is a union, i.e. an OR filter, of all the assertions a particular RMIBManager has requested. Each such EFD is created on a *different* association, with an *empty* destination attribute. The OSIMIS agent realisation of an EFD conceives such a request to imply "*use the association in which you were created to report future events*". If the manager terminates this association without deleting the EFD, the EFD's operational state becomes "disabled". The RMIBAgent is thus able to demultiplex event reports and pass them to the relevant RMIBManager based on the association it receives them. This of course requires that the underlying CMISE API supports explicit association control. Though this approach works, there are two problems with it:

a) the proposed EFD behaviour in the agent is not standard, i.e. not prescribed in [X734], but was simply devised by G. Knight of UCL together with the author; and

b) the manager needs to keep the relevant association open continuously, which results in consuming network resources in connection-oriented networks and increases the memory size of the relevant applications.

It should be noted that despite the fact this approach is not standard, it was pioneered in OSIMIS and was subsequently adopted by many other commercial products. This means that interoperability between those products is possible. On the other hand, an implementation adhering strictly to [X734] should reject the creation of an EFD with no destination attribute value supplied. The problem can be properly solved if the name of the EFD that triggered the event report is added in the m-eventReport primitive as shown in the Code 3-4 caption. This of course requires a revision of the CMIS/P recommendations [X710][X711].

```
EventReportArgument ::= SEQUENCE {
    managedObjectClass      ObjectClass,
    managedObjectInstance   ObjectInstance,
    eventTime               [5] IMPLICIT GeneralizedTime OPTIONAL,
    eventType               EventTypeId,
    eventInfo               ANY DEFINED BY eventType OPTIONAL,
    efdObjectInstance       ObjectInstance -- added parameter
}
```

Code 3-4 Proposed Modification of the CMIS/P EventReport PDU

3.5.3.3 The RMIB API

A part of the API specification for the RMIBAgent and RMIBManager classes is shown in the Code 3-5 caption. A number of customised versions of the same method with different parameters are offered to suit different requirements. For example, the most powerful m-get method allows one to request many attributes from different objects through scoping and filtering in a synchronous or asynchronous fashion. Once though the name of an object is known, simpler methods may be used e.g. a synchronous method to retrieve a single attribute. Requests for asynchronous operations manifest themselves through an argument that passes the "identity" of the relevant RMIBManager in order to be called back.

```
class RMIBAgent : public KS {
    friend class SMIBAgent
    // . . .

public:
    int bind (char* applName, char* host = NULL);
    int unbind ();

    int connect ();
    int disconnect ();

    // most powerful m-get with scope/filter (sync or async)

    int Get (char*objName, char* scope, char* flt, CMISSync sync,
             char* attrNames[], CMISObjectList*& resultList,
             RMIBManager* rmibMgr = NULL, int nByN = 0);

    // synchronous m-get for one object, one attribute only

    int Get (char* objName, char* attrName,
             Attr*& attrVal, AVA*& errInfo = NULLAVAREF);

    // synchronous m-get for one object only, many attributes

    int Get (char* objName, char* attrNames[],
             AVAArray*& attrs, AVA*& errInfo = NULLAVAREF);

    // synchronous m-action for one object only

    int Action (char* objName, char* actionType,
             Attr* actionInfo, Attr*& actionReply,
             AVA* errInfo = NULLAVAREF);

    // request and cancel event reports through string filter

    int receiveEvent (char* flt, RMIBManager* rmibMgr);
    int stopReceiveEvent (char* flt, RMIBManager* rmibMgr);

    // . . .
};

class RMIBManager {
public:
    virtual int EventReport (char* objClass, char* objName,
                     char* eventType, char* eventTime,
                     Attr* eventInfo, Attr*& eventReply,
                     RMIBAgent* rmibAgent);

    // . . .
};
```

Code 3-5 The RMIB O-O API

The Code 3-6 caption shows two usage examples which demonstrate the nature of the RMIB API, which is similar to that of recent object-oriented frameworks such as OMG CORBA. The GDMO/ASN.1 specification of the *uxObj* and *simpleStats* managed object classes used in the examples can be found in Appendix C. We will "walk through" the example and explain what is happening in order to demonstrate how the various features are used.

```
// bind to the application with name "SMA-athena"

RMIBAgent rmibAgent;
if (rmibAgent.bind("SMA-athena") != OK)
    error("cannot bind to SMA-athena!");

// find the name of uxObj instance through scoping/filtering

rmibAgent.Get(NULL, "1stLevel", "(objectClass=uxObj)",
                          NULL, NULL, resultList);
if (resultList->getError())
    error("problem with the uxObj instance at SMA-athena");
char* uxObjName = resultList->first()->getName();
delete resultList;

// request the value of the nUsers attribute

GaugeInt* nUsers; AVA* errInfo;
rmibAgent.Get(uxObjName, "nUsers", nUsers, errInfo);
delete uxObjName;

if (! errInfo) {
    cout << "nUsers is ", nUsers->getint() << endl;
    delete nUsers;
}

// bind to the application "STATSRV" (statistics server)

if (rmibAgent.bind("STATSRV") != OK)
    error("cannot bind to STATSRV!");

// perform a sq. root calculation on the object "statsId=null"

Real arg(4), *res;
rmibAgent.Action("statsId=null","calcSqrt", &arg, res, errInfo);
if (! errInfo) {
    cout << res->getreal() << endl;
    delete res;
}
```

Code 3-6 Example Use of the RMIB Infrastructure

The initial knowledge is that the uxObj class is realised by agents with the name SMA-<host> (SMA stands for System Management Agent). We would like to see how many users are currently logged in at the host "athena". This means the application name we are interested in is "SMA-athena". Note that this is not a violation of location transparency since we do not specify the location where that application runs. It might *not* run at host athena but communicate with through another protocol i.e. be a Q-Adapter in TMN terms. We first instantiate a RMIBAgent and bind through it to "SMA-athena". The RMIBAgent knows the local domain and will construct the directory name for the SMAP object of that application as explained in Chapter 2 e.g. `c=UK@o=UCL@ou=CS@cn=SMA-athena`. It will then contact the directory, retrieve the presentation address of the contained SMAE object and try to connect to that address. If any of those remote operations fails, NOTOK will be returned from the bind method and the program subsequently will exit through *error*. If an association is established, it will be "cached" for 60 seconds which is the default caching period.

Now we are bound to that system and would like to retrieve the *nUsers* attribute of the *uxObj* instance, but we do not know its name. We know though from its GDMO specification that it should be in the first level of the MIT as it is "bound" to the class *system* [X721] which is always at the top of the MIT. We then perform then a m-get operation to the top MIT object whose LDN is "empty", request the first level subordinates, apply the filter *(objectClass=uxObj)* and request no attributes. This will result in two CMIP PDUs passed back to the RMIBAgent, one with the uxObj instance information and one being the empty terminator (note that scoping results in linked replies even if one object only is selected). This request took place in a synchronous fashion and we now know the name of the uxObj instance. We subsequently perform another synchronous m-get operation to the latter, request the nUsers attribute and print its value. We could have retrieved the nUsers attribute with the first operation which would have avoided the second one, but we are trying to demonstrate more aspects of the RMIB API.

We subsequently bind to another application, called STATSRV, which contains an instance of the simpleStats class. The RMIB agent terminates first the connection to SMA-athena which is still established and then goes through the same process of constructing the directory name of the application, retrieving its address and connecting to it. It should be noted that the connection to the directory server exists already as it has been "cached-in" by the underlying infrastructure in a similar fashion. In the second example we know in advance the name of the simpleStats instance which is simpleStatsId=null. The latter is a convention used for single-instance classes i.e. the value of naming attribute to be *"null"*. We subsequently perform the "calcSqrt" action with argument 4 and print the result, which should be 2.

The above example, though simple, demonstrates the power and simplicity of the RMIB infrastructure. It should be noted that the O-O ASN.1 API contributes significantly to the relevant power and simplicity. Finally, it should be stated that this is a "weakly-typed" environment, where type mismatch errors are discovered only at run-time. For example, if the argument to the calcSqrt action was defined as "GraphicString arg("4")", the program would compile happily but the action method call would return NOTOK when validating the consistency of the arguments and would print a diagnostic error message. If, the reply was specified of type GraphicString, the program would compile happily, perform the action fine but it would either crash when trying to print the type or it would simply print garbage.

In an agent-level infrastructure such as the RMIB it is *impossible* to support strong typing since the relevant methods operate still at the CMISE level and precise ASN.1 types are not known. This observation brings us naturally to the next level of management infrastructure that operates at the managed object level and can be strongly-typed, the Shadow MIB.

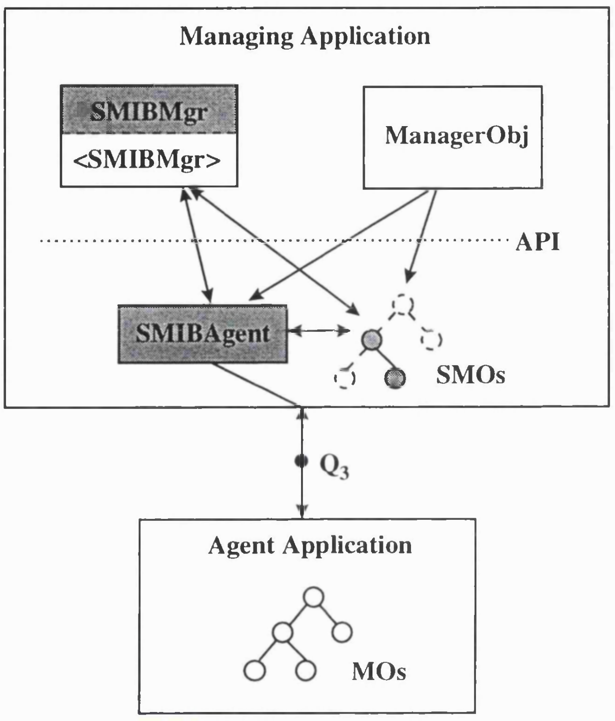
3.5.4 The Shadow MIB Managed Object Level Infrastructure

The idea for a facility that "caches" managed objects in manager applications was conceived early in the NEMESYS project, before the use of OSIMIS and TMN Q_3 protocols. At the time (1988-1989), Objective C [Cox86] was used which supports an object serialisation facility in a similar fashion to Java [Sun96]. The first NEMESYS management platform used Sun RPC [Sun88] to communicate management information in the form of serialised managed objects or simply serialised attributes, operation parameters and results. Based on this "management protocol", an application infrastructure was developed, known as the Management Unit Information Base (MUIB). This provided support for remote operations through local operations on "cache" objects. The reader may observe the similarity of this approach to the much more recent Java Remote Method Invocation (RMI). The main contributor of this idea and implementer of the relevant infrastructure was P.-E. Stern of GSI Erli, France.

When OSIMIS was introduced in the last phase of the NEMESYS project, the author thought of reproducing the MUIB functionality over true Q_3 interfaces in order to support novice application developers by hiding the details of Q_3. The concept was named Shahow MIB (SMIB) and a detailed specification was produced, following a weakly-typed approach. The concept was embraced with enthusiasm by IBM ENC, Heidelberg, who were a NEMESYS partner, and D. Jordaan provided an embryonic OSIMIS-based implementation of the initial SMIB specification as a proof of concept. This was not used in NEMESYS but IBM ENC took the concept further a few years later, resulting in the Object-Oriented Interface (OOI) specification [Holb95] which influenced the NMF GDMO/C++ API [Chat97]. The author independently explored the concept further during 1993, together with A. Carr of Cray Communications, UK, in the context of the RACE ICM project. The relevant issues were revisited and a new specification was produced and implemented in OSIMIS by A. Carr. It should be mentioned that the SMIB infrastructure was never released with OSIMIS since it was more of an experimental prototype rather than robust infrastructure for building TMN applications.

3.5.4.1 The SMIB Model

The key characteristic of the SMIB approach is that it supports a local cache of (parts of) the remote MIB within the manager application in the form of Shadow Managed Objects (SMOs) [Pav94b]. These are administered by a Shadow MIB Agent (*SMIBAgent*) which models the "binding" to that system, in a similar fashion to the RMIBAgent. In fact, the SMIBAgent contains a RMIBAgent instance which it makes available to its users. This means that the SMIB framework is a clear superset of the RMIB one. The difference between this approach and the IBM OOI or the NMF GDMO/C++ is that the RMIB and SMIB infrastructures are distinct, so that the RMIB infrastructure can be used *without* the SMIB. In those frameworks, the relevant concepts are tightly-coupled and, as such, inseparable.



<SMIBMgr> - specific class derived from SMIBMgr

Figure 3-10 Shadow MIB Model and Interactions

The Shadow MIB model is shown in Figure 3-10, in the form of co-operating object instances. The API can be both synchronous and asynchronous, with the *SMIBManager* class providing the abstract callback interface. It should be emphasised that only the MOs the application is interested in are shadowed; these are shown in shaded form in Figure 3-10. SMOs can be created either explicitly, through a request to the SMIBAgent with their name, or implicitly as a result of a scoped request either to the SMIBAgent or to another SMO. The SMOs typically hold information regarding the name, class and attributes of the "master" MO in the agent. Note that the SMIBAgent encapsulates and uses an instance of RMIBAgent, which provides O-O access to the real remote agent. The RMIBAgent instance is not shown in Figure 3-10 but this relationship is shown in the following figure.

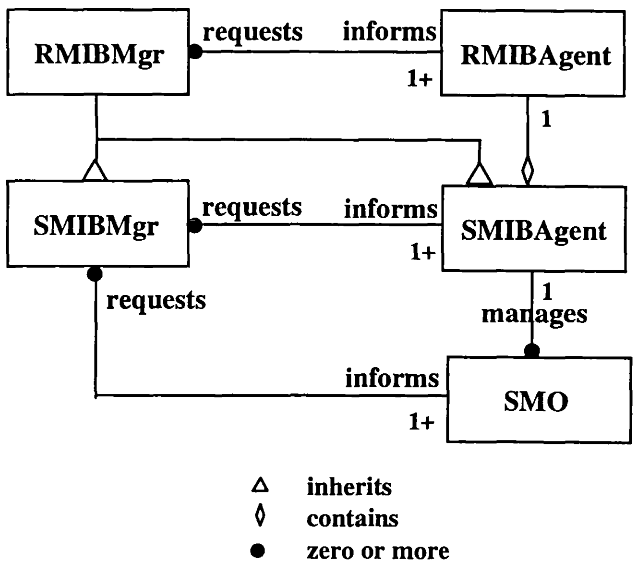


Figure 3-11 OMT Relationships of the RMIB and SMIB Classes

Figure 3-11 shows an OMT diagram of both the RMIB and SMIB models. Relationships other than inheritance and containment are explicitly named while instance relationships include their cardinality. The SMIBAgent contains one RMIBAgent and it is itself a RMIBManager in order to receive events and asynchronous results. The SMIBManager derives from the RMIBManager so that it can receive events and asynchronous results both from the SMIBAgent and the contained RMIBAgent (the SMIB model is a clear superset of the RMIB one). The SMIBAgent manages a number of SMOs which are members of the relevant group. Finally, the manager objects *request* information from the agent and shadow objects and are *informed* of events and asynchronous results.

Manager objects access typically the SMOs either in a local fashion, in order to retrieve values of cached attributes, or in a fashion that triggers a remote operation to the associated master MO (m-get, m-set, m-action). The creation and deletion of SMOs can also take place in two modes: a local mode, which does not affect the master MO, and a remote or real mode, which results in the creation and deletion of the master MO through m-create and m-delete. Operations with scope and filter can be performed either to the SMIBAgent, using explicitly the base object's name, or to an SMO, which in this case assumes the role of the base object. In the same fashion, event reports may be requested either from the SMIBAgent, when a general filter assertion is specified that spans different object classes and instances, or from a particular SMO, requesting events either from the individual instance or from all the instances of that class.

There are two different approaches for the design of an SMIB framework supporting the model described above:

- a weakly-typed approach, in which a single SMO class is used to model any managed object class in the remote agent; and
- a strongly-typed approach, in which specific SMO classes are produced for the corresponding MO classes through a GDMO compiler.

The weakly-typed approach is suitable for generic applications such as MIB browsers and its main advantage over the RMIB is that it provides a cache for the MO's attributes, alleviating the task of the application developer. Individual attributes can be requested to be updated at regular intervals, according to some predetermined schedule or through attributeValueChange and stateChange event reports if the MOC supports those.

The strongly-typed approach adds the advantage that attribute access methods and actions are strongly-typed. In addition, since the classes are produced through a GDMO compiler, the user may add methods for adaptive polling or other behaviour according to the application's management policy and the semantics of the particular object class.

3.5.4.2 The SMIB API

The Code 3-7 caption shows an example of using the SMIB infrastructure. The first part demonstrates the weakly-typed, generic approach which is very similar to the RMIB one. The second part demonstrates the strongly-typed, information model specific approach, supported through a GDMO compiler. The example is similar to the first one used in the RMIB case, i.e. retrieve the nUsers attribute from the uxObj instance, but we now also perform an "echo" action to that instance (the full class specification can be found in Appendix C). Incidentally, the echo action demonstrates how to say "hello world" in an OSI-SM/TMN distributed fashion!

After the SMIBAgent is instantiated and bound to the SMA-athena application, we request the creation of a SMO with name "`uxObjId=null`". In the general case, the SMIBAgent will transparently send a m-get request to the master object, retrieving its objectClass, allomorphs and packages attributes i.e. its *top* class [X721] part, in order to verify that the SMO is valid and populate it with the necessary "base" information. The second boolean argument in our call though indicates that we would also like to retrieve at the same time all its attributes, which means that the SMIBAgent will request all the attributes in this case. Having succeeded, we retrieve the nUsers attribute locally, which demonstrates the use of the shadow object as an attribute cache. We subsequently perform the echo action. You may note that the latter is extremely similar to the equivalent RMIB method in the Code 3-6 caption. In fact, there is no real advantage in the SMIB approach over the RMIB one regarding actions, attribute sets and object creations / deletions. The key advantage is attribute value caching and relevant generic update policies which can be encapsulated in the SMO class realisation.

```
// weakly-typed, generic SMIB approach

SMIBAgent smibAgent;
smibAgent.bind("SMA-athena");

AVA* errInfo;
GraphicString arg("hello world"), *res;
Gauge* nUsers;

SMO* smo = smibAgent.getSMOHandle("uxObjId=null", True);

if (smo) {
    attr = (Gauge*) smo -> getAttr("nUsers", False);
    smo -> Action("echo", &arg, res, errInfo);
    // . . .
}

// strongly-typed, information model specific SMIB approach

SMIBAgentUx smibAgentUx;
smibAgentUx.bind("SMA-athena");

uxObjSMO* uxObjSmo = smibAgentStats.uxObjFactory("uxObjId=null");

if (uxObjSmo) {
    nUsers = smo -> get_nUsers(False);
    uxObjSmo -> echo(arg, res, errInfo);
    // . . .
}
```

Code 3-7 Example Use of the SMIB Infrastructure

The strongly-typed approach requires a GDMO compiler that will produce specific SMO-derived classes, e.g. uxObjSMO, and also a specific SMIBAgent for the particular information model. The latter is necessary in order to include strongly-typed methods for getting access to the relevant specific shadow objects, as demonstrated in the example. There will typically exist one specific SMIBAgent class for every information model the manager "knows" about e.g. SMIBAgentG774 for the SDH element model [G774]. Returning to the example, we bind to that system and request the creation of a shadow object in the same fashion as before. The key difference is that the relevant method (uxObjFactory) is now strongly-typed. The same is true for the subsequent attribute retrieval (get_nUsers) and the echo method call. For example, if the echo argument or result was declared of type Real by mistake, the program would not compile. This approach has significant advantages for program development by novice users.

3.5.4.3 Manager Mapping of GDMO to O-O Programming Languages

The SMIB approach requires essentially a "manager" mapping of the GDMO O-O abstract language to concrete O-O programming languages. We have implicitly proposed aspects of such a mapping while discussing the SMIB model and explaining its features by example. We elaborate on this mapping and present an explicit proposal below:

- Managed object classes map to C++ classes. In the weakly typed approach, *every* GDMO class is modelled by the SMO class. In the strongly-typed approach, GDMO classes map to corresponding C++ classes with the *SMO* suffix, following exactly the same inheritance relationships as in GDMO. The *topSMO* class derives from SMO.

- Packages are not explicitly present in shadow managed objects. Their attributes, actions and notifications map onto methods of the containing managed object class as proposed below. These methods are produced for both mandatory and conditional packages. If a conditional package is not present in the MO instance, the associated SMO methods to that package should return an error.

- Attributes map to relevant access methods according to the access rights, i.e. *get*, *set*, *setToDefault*, and also *add* and *remove* for settable multi-valued attributes. Access could be either local or remote. Methods should be provided to allow the remote get and set of more than one attributes at a time i.e. resulting in one CMIS m-get or m-set request. In the strongly-typed approach, specific methods should be generated for each attribute with the exact ASN.1 type e.g. "Gauge* nUsers_get()", "int wiseSaying_set(GraphicString*), etc.

- Actions map to relevant access methods, with an input argument modelling the GDMO "action information" and an output argument modelling the GDMO "action reply". In the strongly typed approach, specific methods should be generated for each action with the exact ASN.1 types e.g. "int echo(GraphicString*, GraphicString*&)".

- In the weakly-typed approach, the SMIBManager provides a generic *EventReport* method, inherited from the RMIBManager as in the Code 3-5 caption. This method is used to "push" a notification to the RMIBManager. In the strongly-typed approach, notifications should be mapped to generated methods of a specific SMIB manager class for that information model. The notification methods should take event information and reply parameters with specific ASN.1 types.

- Parameters for attributes, actions and notifications indicate essentially MOC-specific errors. These should be mapped onto error parameters, with type-value information for the relevant methods (OSIMIS maps those to AVA error arguments).

- Name bindings do not map explicitly to shadow managed objects. Every SMO knows its name and, as such, its position in the MIT. SMOs should provide facilities for getting the handle of the superior SMO or the handles of subordinate ones through scoping and filtering.

The proposed mapping highlights a "failing" of GDMO as an abstract O-O specification language. Arbitrary methods of distributed managed objects are modelled in GDMO through actions. These take only one input argument, the action "information", and return one output argument, the action "reply". In the case of actions with more than one input or output arguments, they have be artificially combined into *one* argument using typically the ASN.1 SEQUENCE type. This results in a non-natural mapping of the arguments to C++ method signatures. As an example, the simpleStats class supports a calcMeanStdDev action which calculates the mean and standard deviation given a list of real numbers (see Appendix C). The action reply in this case should comprise two distinct output arguments, the *mean* of type REAL and the *stdDev*, again of type REAL. This is not possible in GDMO, so one is forced to define a new output argument of type MeanStdDev, which is of type SEQUENCE and comprises two elements of type REAL for the mean and standard deviation. The "contrived" ASN.1 type and the resulting action signature in C++ are shown in the Code 3-8 caption, together with the ideal method signature for this type of action.

```
-- ASN.1 definition for "calcMeanStdDev" action reply

MeanStdDev ::= SEQUENCE {
    mean        REAL,
    stdDev      REAL
}

// resulting action signature in C++

int calcMeanStdDev(RealList* info, MeanStdDev*& reply);

// ideal action signature for the "calcMeanStdDev" action

int calcMeanStdDev(RealList* rList, Real*& mean, Real*& stdDev);
```

Code 3-8 The C++ Mapping "Problem" with GDMO Actions

The solution is to modify the GDMO Action template to allow for more than one information and reply types. In this case, the action and reply information at the CMIS/P level will be a SET OF ActionInfo and a SET OF ActionReply respectively. This implies a modification of CMIS/P in addition to GDMO. The Code 3-9 caption depicts both the grammar for the new template and an example for the calcMeanStdDev of the simpleStats class. The proposed syntax would result in the more natural C++ signature for that action as explained above.

```
<action-label> ACTION
    [MODE CONFIRMED ;
    ]
    [PARAMETERS         <parameter-label>
                      [,<parameter-label>]* ;
    ]
    [WITH INFORMATION SYNTAX
                        [<argument-label>] <type-reference>
                      [,[<argument-label>] <type-reference>]* ;
    ]
    [WITH REPLY SYNTAX
                        [<result-label>] <type-reference>
                      [,[<result-label>] <type-reference>]* ;
    ]
REGISTERED AS object-identifier ;


calcMeanStdDev ACTION
    MODE CONFIRMED;
    WITH INFORMATION SYNTAX
        numbers UCL-ASN1Module.RealList;   -- SET OF REAL
    WITH REPLY SYNTAX
        mean    UCL-ASN1Module.Real,       -- REAL
        stdDev  UCL-ASN1Module.Real;       -- REAL
REGISTERED AS { uclAction 702 };
```

Code 3-9 Proposed Modification of the GDMO Action Template

A weakly-typed SMIB approach was prototyped during 1994 in the ICM project, validating the design and GDMO to C++ mapping presented above. The reason the weakly-typed approach was chosen was practical: the strongly-typed approach needs a GDMO compiler with an SMIB-specific back-end for code generation. At that time (early 1994), the OSIMIS GDMO compiler was not entirely stable. The SMIB prototype was incomplete, so it was never released with OSIMIS. On the other hand, this implementation validated the relevant concept and model.

3.5.5 The Tcl-RMIB Scripting Manager Infrastructure

The Tool Command Language (Tcl) [Oust94] emerged in 1994 as a general-purpose scripting language that interfaces well with C/C++ and can be extended with commands implemented in those languages. The key advantage of Tcl is its interpreted nature which accelerates the development process and may also support code mobility. Tcl is a weakly typed language based on strings and it is relatively easy to use compared to compiled programming languages such as C/C++. On the other hand, it is about an order of magnitude slower compared to those.

One of the main reasons behind the popularity of Tcl is its extension for the MIT X Window System and MS Windows, known as Tk [Oust94]. This is a graphical toolkit that extends the core Tcl facilities with additional commands for constructing GUIs. The Tk toolkit exists in the form of a collection of display "widgets", providing a user-friendly way to compose graphical primitives. The combination of Tcl and Tk presents a suitable environment for the rapid construction of GUI-based applications. A Tcl extension of the compiled higher-level manager infrastructures could support the rapid construction of TMN WS-OS applications.

We have chosen to provide RMIB extensions to Tcl since RMIB is the primary high-level manager infrastructure. Providing those Tcl extensions was relatively straightforward since the RMIB infrastructure supports already string parameter passing for distinguished names, filters, and attribute, action, notification and specific error values. A more difficult aspect is the mapping of the object-oriented RMIB model to the procedural Tcl one. Given the fact Tcl allows to reference and invoke procedures in a similar fashion to the C language "function pointer" facility, it is possible to emulate object-oriented style of programming in Tcl. [Meyer88] discusses how object-oriented features can be supported in procedural languages that support a facility equivalent to "function pointers".

The resulting language is called Tcl-RMIB [Tin95][Pav96d] and allows interactions with remote agents using the same abstractions as the compiled C++ RMIB. Tcl-RMIB provides a number of extension commands to Tcl in which the underlying RMIB C++ objects are manipulated by their interpreted counterparts. Commands are provided for the user control (creation, deletion, etc.) of the agent and manager objects locally in Tcl and for performing management operations through those objects. Both synchronous and asynchronous modes of operation are supported. The management commands are shown in Table 3-5. Their syntax owes much to that of the generic command line manager programs which also influenced the string-based CMIP that was described in section 3.3.2.4.

Management Command	Description
m_bind *agentId* ?-a *agentName*? ?-h *hostName*?	binds to a remote agent
m_unbind *agentId*	unbinds from a remote agent
m_get *agentId* ?-c *class*? ?-n *name*? ?-s *scope* ?*sync*?? ?-f *filter_expr*? ?-a *attribute* ...? ?-m *managerId ?-o*??	allows the M-Get invocation; may return an invoke identifier for call-back correlation
m_set *agentId* ?-c *class*? ?-n *name*? ?-s *scope* ?*sync*?? ?-f *filterExpr*? ?-w|d|a|r *attrType*?=*attrValue*? ...? ?-m *managerId*?	allows the M-Set invocation; may return an invoke identifier for call-back correlation
m_action *agentId* ?-c *class*? ?-n *name*? ?-s *scope* ?*sync*?? ?-f *filterExpr*? ?-a *actionType*?=*actionValue*?? ?-m *managerId*?	allows the M-Action invocation; may return an invoke identifier for call-back correlation
m_delete *agentId* ?-c *class*? ?-n *name*? ?-s *scope* ?*sync*?? ?-f *filterExpr*? ?-m *managerId*?	allows the M-Delete invocation; may return an invoke identifier if call-back correlation
m_create *agentId* ?-c *class*? ?-n *name* | -s *superiorName*? ?-r *referenceName*? ?-a *attrType*=*attrValue* ...? ?-m *managerId*?	allows the M-Create invocation; may return an invoke identifier for call-back correlation
m_notify *agentId managerId* ??-c *class*? ?-n *name*? ?-e *eventType*? | ?-f *filterExpr*? | ?-a?? ?-s?	allows the call-back registration of a manager to receive event reports and also request termination of notification; event reporting may be cancelled (using -s).

Table 3-5 Tcl-RMIB Management Commands

The Tcl-RMIB approach has semantics of a weakly-typed dynamic invocation interface, in a similar fashion to the RMIB one. It mirrors the functionality of the compiled RMIB approach and provides an analogous interpreted interface. The full CMIS expressive power is available i.e. scoping, filtering, linked replies and fine-grain event reporting based on filtering. Control over management associations is provided while it may be also left to the underlying infrastructure. A generic list structure that applies to all the requests is used for replies and errors as described in detail in [Tin95].

While the Tcl-RMIB approach is agent-oriented in a similar fashion to the RMIB one, higher-level managed object-oriented interpreted approaches are possible e.g. a Tcl-SMIB. It should be mentioned that while Tcl was thought to be the definitive interpreted scripting language 2-3 years ago, it has been recently overshadowed by the emergence of Java [Sun96]. The latter has all the advantages of Tcl/Tk, including facilities for rapid GUI construction. In addition, it is object-oriented, it compiles into intermediate "byte-code" representation which results in better performance and will be eventually supported by Java-capable hardware. In summary, Java is the language to be used for TMN WS-OSs in the future. A Java-RMIB infrastructure is perfectly feasible to provide since Java can interface to C++ (though not as easily as Tcl). In addition, the fact that Java is object-oriented and has syntax similar to C++ will result in exactly one-to-one mapping between the Java-RMIB and the RMIB features while equivalent methods will have similar syntax.

The Tcl-RMIB infrastructure was designed together by T. Tin and the author while it was implemented by T. Tin in early 1995. It was subsequently used successfully in the ICM project for developing WS-OS applications. It was also publicly released with OSIMIS-4.0 [Pav95b], so it was also used by the wider community. It demonstrates that the concepts and facilities of the high-level manager infrastructure are general enough to allow mappings to programming languages other than C++. In addition, it suggests that interpreted high-level manager support in languages with built-in GUI capabilities is an important ingredient of a TMN distributed software platform.

3.5.6 The Management Information Repository

While discussing both CMISE and higher-level manager infrastructures and APIs, we referred to necessary knowledge of the GDMO information model. This is required in order to be able to map user-friendly string names of attributes, actions, events and specific error names to the respective object identifier and the corresponding syntax. This information should be made available to such infrastructures in a data-driven fashion so that they are able to cope with new information models without changes in their code. As such, it needs to be stored in a form of database that can be accessed at run-time, hence the term *management information repository*.

This information is typically produced by compiling a GDMO information model and has to be added to the information repository if it is not already there. This process can be automated so that every time an information model is compiled, the repository is automatically updated. A manager infrastructure, such as a high-level CMISE implementation or the RMIB, typically reads this information when starting up and builds-up a core memory image of the relevant mappings. Access to this information is provided either through the name or the associated OID, which serves as the "key". A typical realisation of this information is through a hash table in order to support fast access to the relevant data.

With this information in place, an attribute name such as the "wiseSaying" of the uxObj class can be mapped onto the 2.37.1.1.7.503 OID and the GraphicString corresponding syntax. This allows, for example, to check the consistency of an attribute value assertion that the user constructs in order to set the value of this attribute. In addition, it allows the infrastructure to map the attribute name to the corresponding OID. In the case of indications, it allows the infrastructure to map the OID to the user friendly name and to construct the correct C++ type with the value i.e. GraphicString in this case.

This is the minimum information required for supporting infrastructures such as the RMIB and the weakly-typed SMIB. Additional information about the GDMO objects may also be part of the repository, supporting further facilities. For example, it might be desirable to perform additional checking in a manager application, before a request is forwarded to the agent. If, say, a manager tries to perform a calcSqrt action on a uxObj instance which the latter does not support, it should be possible to detect this locally and generate an error. This can only be achieved if the repository contains additional information about managed object classes e.g. the packages, attributes, actions, notifications and specific errors they support. This information is typically referred to as *meta-data* or *meta-information* since it describes the information model itself.

The key benefit of having access to this type of information in manager applications is that requests can be validated locally, before sending them to the agent. An additional benefit has to do with generic applications such as MIB browsers. In those applications, it is possible to present this meta-information to the user in order to support better a particular request. For example, when creating a new object instance, the user may be presented with a list of those attributes that may be initialised so that s/he may supply initial values. If this information is not available, the human users need to have the GDMO model "in their head", which is certainly not possible.

This meta-information can be produced through the GDMO compiler, stored in the repository and read by manager applications at start-time in order to build an image in core memory. Incidentally, the same type of information is required in agent applications. The core memory representation of this type of information can be a tree of meta-class objects that model accurately the GDMO inheritance relationships, together with a container object that provides access to those through the class name or OID.

```
# name              OID
uxObj               uclManagedObjectClass.50
uxObj-system        uclNameBinding.50
uxObjPackage        uclPackage.50

# attr name         OID                   attr type
uxObjId             uclAttributeID.501    SimpleNameType
sysTime             uclAttributeID.502    UTCTime
wiseSaying          uclAttributeID.503    GraphicString
nUsers              uclAttributeID.504    ObservedValue

# action name       OID                   info type           reply type
echo                uclAction.502         GraphicString       GraphicString
```

Code 3-10 Structure of the OSIMIS Management Information Repository

The approach followed in OSIMIS was that of the minimum possible meta-information i.e. the table mapping names to OIDs and ASN.1 types. This means that it is not possible to detect "mismatch" errors locally, within manager applications. Requests are always sent to the agent that will detect the error. This is acceptable, since such errors are typically development type errors hat will be sorted out before the application is deployed in a real system. The real drawback is that generic OSIMIS applications such as the MIB browser [Pav92a] do not have access to class information that could be used to provide a friendlier interface as describe above.

Finally, the Code 3-10 caption shows the OSIMIS structure of the management information repository using as an example the uxObj class which is formally specified in Appendix C. There are *three* types of "database records": those for classes, name bindings and packages which do *not* have an associated syntax; those for attributes that have *one* associated syntax; and those for actions and notifications that may have *two* associated syntaxes, one for the information and one for the reply. Note that the OID prefixes are also part of the database so that OIDs are fully resolved. Note also that the objectCreation, objectDeletion and attributeValueChange notifications, which the uxObj class supports, are absent. This is because these are generic ones, introduced by the Object Management SMF [X730] and specified in [X721]. As such, they are already in the X.721 part of the repository. This means that the repository is constructed in a modular fashion.

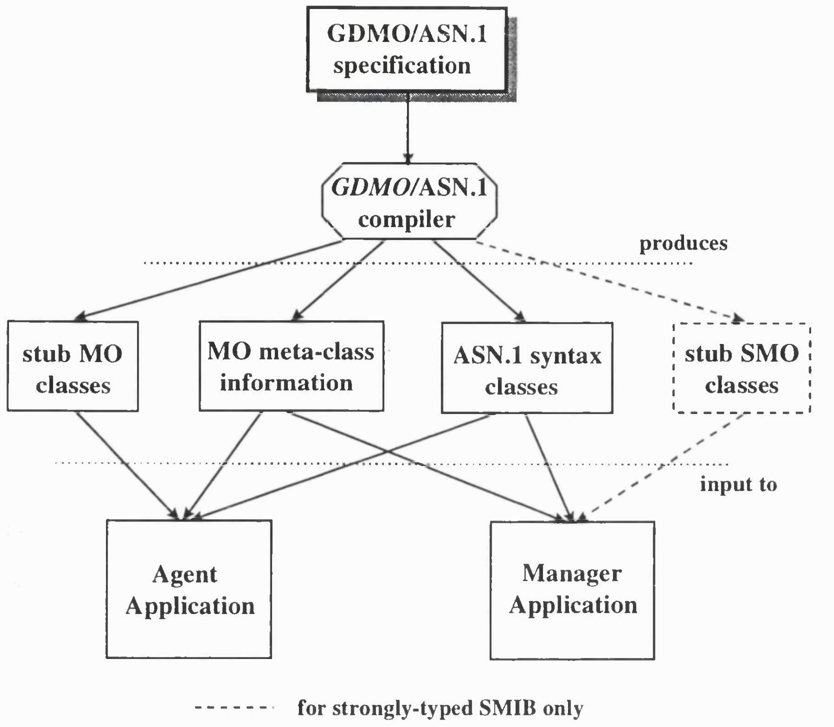


Figure 3-12 Information Produced Through the GDMO/ASN.1 Model

We will close this section with a note on realisation aspects and procedures. While these have been pioneered in OSIMIS, they are also followed by most commercial TMN platforms. Every time an agent application with a new management information model is implemented, the following steps are followed. The GDMO/ASN.1 model is compiled first in order to produce the "stub" C++ managed object classes to support the realisation of the agent part, as it will be described in section 3.6. The compilation of the ASN.1 part will result in C++ classes for the new ASN.1 types this information model introduces, produced through the O-O ASN.1 compiler. The managed object classes and associated syntax classes need to be kept separate, since the syntax classes are used by both agent and manager applications. If the relevant TMN platform supports a strongly-typed SMIB infrastructure, shadow object classes will be also produced. Finally, the meta-class information will be produced for the management information repository. The ASN.1 classes and the management information repository are used by both agent and manager applications, as shown in Figure 3-12. Note that the information model which is "built-in" in both the agent and manager logic constitutes the shared management knowledge which together with the supporting Q_3 protocol stack achieves interoperability, as discussed in Chapter 2.

High-level manager infrastructures will need to be made "aware" of the new ASN.1 syntaxes. In software terms, this means that the relevant library should be linked with the application and that the RMIB code should be "told" somehow of the new syntaxes. In OSIMIS the latter involves changing *one* line of code in a header file in order to include another header file produced by the O-O ASN.1 compiler. This is not an issue for specific manager applications since these will be implemented "from scratch" with information model specific logic and behaviour. Generic manager applications though, such as an MIB browser [Pav92a], will need to be slightly modified and re-linked with a new syntax library every time an information model with new syntaxes is encountered.

While this is acceptable in most cases, there may exist situations in which it is not desirable to stop a generic manager application, re-link it with new syntaxes and restart it. Commercial TMN platforms support facilities so that new ASN.1 syntaxes can be incorporated "on the fly". This is done through additional meta-data produced by the ASN.1 compiler for every syntax which "guide" the application how to *print* and *parse* relevant instances. In this case, whenever new names and types are encountered, the generic application needs to read again the information repository in order to "learn" how to manipulate those types. This type of behaviour is possible but introduces additional complexity. The author thought of accommodating this feature when designing the OSIMIS ASN.1 API but decided against it because of its complexity. The solution adopted in OSIMIS has an impact on generic applications which cannot deal with new information models and syntaxes in a fully dynamic fashion. On the other hand, this is acceptable in most real-life situations.

3.5.7 Summary

In this major section we looked at issues behind realising high-level manager infrastructures and APIs that hide the complexity of the underlying CMIS service without sacrificing any of its expressive power. We identified two types of abstractions and relevant APIs:

- weakly-typed APIs that model a remote agent application; the relevant OSIMIS infrastructure is known as the Remote MIB (RMIB); and
- both weakly-typed and strongly-typed APIs that model individual managed objects in a shadow or proxy fashion; the relevant OSIMIS infrastructure is known as the Shadow MIB (SMIB).

Both these infrastructures have a lot of commonality with emerging distributed object frameworks such as CORBA [CORBA] and DCOM [DCOM], providing an easy to use APIs and supporting access and location transparencies. Some of the relevant features were demonstrated through examples in the Code 3-6 and Code 3-7 captions. A few lines of code are enough to perform operations on remote managed objects, in a similar fashion to emerging distributed object frameworks. In addition, the CMIS scoping, filtering and fine-grain event-reporting power is available through the same simple APIs. This is the major contribution in this section.

Having examined aspects of the RMIB and SMIB infrastructures in detail, we conclude here that the right approach for the SMIB is the strongly-typed one, which has serious advantages over the RMIB approach. Strong-typing can be combined with user-included behaviour in the relevant shadow classes, alleviating the task of the management application developer. The weakly-typed SMIB approach is syntactically very similar to the RMIB one as demonstrated through the examples. In addition, the generic "caching" behaviour of that approach is only of limited usefulness, since it can not exploit the semantics of the management information.

We have also shown how to map those infrastructures to interpreted scripting languages with "integrated" GUI development support such as Tcl/Tk. The latter though has been rendered "obsolete" by the emergence of Java, which is now the prime candidate for TMN workstation development. Since Java is an object-oriented language, it would be natural to map to it the RMIB and SMIB models presented in this section.

In summary, we demonstrated in this section how to provide an object-oriented platform infrastructure for the manager part of the OSI-SM/TMN manager-agent model. As a result of the relevant research towards providing such infrastructures, we identified weak aspects in the CMIS/P and GDMO specifications for which we proposed relevant solutions. The RMIB and SMIB models and APIs have been input to the NMF TMN/C++ effort [Chat97]. This proposes a industrial solution to the problem of TMN high-level manager APIs which has a lot of commonality with the models presented in this section.

3.6 Issues in Realising Object-Oriented Agent Infrastructures

3.6.1 Introduction

In the previous section we concentrated on issues related to the realisation of object-oriented infrastructures for applications in manager roles. The key issue was to provide object-oriented abstractions of whole agent applications or managed objects, giving the illusion that these objects were available in the local address space. The result was user-friendliness and easy programmability that hides underlying protocol aspects; this is an important attribute of distributed object frameworks as identified in section 3.2.2.

A similar infrastructure is required for applications in agent roles. In this case, the behavioural aspects of managed objects should be shielded from CMIS/P access aspects as much as possible, allowing designers and implementors to concentrate in the intelligence of applications rather than been concerned with the underlying access details and complexity. A key issue in such an infrastructure is the mapping of the GDMO abstract language to a concrete object-oriented programming language such as C++.

Other important issues in realising agent infrastructures are the following: support for CMIS access aspects such as name resolution, scoping, filtering and linked replies; support for the allomorphic behaviour of object instances; support for different models in maintaining the consistency of managed object attributes and associated resources; support for managed object persistency; and support for the systems management functions and in particular for event reporting and logging.

The realisation of object-oriented agent infrastructures was an issue that attracted significant attention from the research community. The author pursued early research in this direction which resulted design and implementation of the OSIMIS Generic Managed System (GMS) [Pav91a]. Related research work is presented in the next sub-section, while the following sub-sections discuss the relevant issues and present a concrete proposal for a generic agent infrastructure.

3.6.2 Related Work

The author's research work in realising object-oriented agent infrastructures preceded other research work in this area. The fundamental principles behind agent infrastructures were presented in [Pav91a] and were validated by the early implementation of the OSIMIS Generic Managed System (GMS). This work was enhanced and refined over the years and has been subsequently presented in more detail in [Pav93a], [Pav95a] and [Pav96b]. Related research work in this area is described in chronological order below.

[Nakai91] proposes a GDMO MIB editor tool which acts also as GDMO/ASN.1 compiler, producing a *data dictionary* for managed object classes. This is stored in a relational database and it is parsed into core memory in order to validate the consistency of operations to managed objects. It forms part of the agent process, which also contains a "protocol processor" and the managed objects.

[Nakak91] proposes an agent infrastructure which contains the following modules: the protocol processing module, the MIT module, the MIB access module and the managed object module. The interesting aspect of this work is the separation of the MIT representation from the managed objects themselves; this approach which has been later adopted by a number of products. A GDMO/ASN.1 compiler called MINT is used to parse specifications and produce stub class definitions. The programming language used is *superC*, a proprietary object-oriented extension of the C language.

[Newn92] describes an agent object-oriented infrastructure in C++ which is built over the XOM/XMP C-based API. This resembles a lot to the OSIMIS managed object stubs. The realisation of the top class provides an API to the "object manager", which performs the CMIS/P functions. It also defines the get, set and action methods in polymorphic fashion, i.e. as C++ virtual methods, so that derived classes can redefine them and add behaviour. The proposed design was validated through a prototype.

[Doss93] describes the design of an agent prototype which is driven by a GDMO/ASN.1 compiler. The internal decomposition of the agent functionality is presented only briefly but the mapping of GDMO to C++ is presented in some detail. C++ classes model the GDMO classes and packages while C++ classes model the ASN.1 attribute, action and notification types. Class specialisation is supported through "class doubling" i.e. behaviour is not added to stub classes produced by the GDMO compiler but a new class needs to be derived from the produced one. The approach is similar to the CORBA Interface Definition Language (IDL) to C++ mapping. Aspects of this work were incorporated in Alcatel's ALMAP TMN platform.

[Deri95] discusses in detail the mapping of the various GDMO templates to C++ classes. It also discusses how filtering assertions can be supported by a generic attribute class, similar to the Attr class presented in section 3.4. The approach reflects the authors' experience from developing various commercial and public domain OSI-SM infrastructures. It is quite similar to the OSIMIS GMS apart from the fact that conditional packages are modelled by separate C++ classes. The acknowledgements state that "*the importance of the OSIMIS platform is acknowledged, since it has greatly contributed to the diffusion of OSI-SM and has introduced a number of concepts now adopted by many commercial implementations*".

[Flau95] discusses the object-oriented aspects of DEC's TeMIP TMN platform. A particularly interesting aspect is that the management applications themselves can be distributed, based on a proprietary "object broker" model which supports a dynamic invocation interface and offers "intra-application" location transparency. The whole framework is data-driven through dictionaries so that new functionality can be added on the fly.

[Feri96] describes IBM's Netview TMN platform and its support environment for hybrid agent/manager applications e.g. TMN OSs. Its key feature is the MIBcomposer front-end GUI. This serves as GDMO/ASN.1 editor and compiler but can be also used to associate behaviour to managed object classes. The agent environment is modular and the functionality of new classes can be introduced without having to shutdown the agent for a "cold start".

[Chat97] describes the NMF GDMO/C++ API in agent role. The MIT is kept separate from the managed objects so that parts of an agent application may be distributed. Behaviour is added through separate classes which are derived from the GDMO produced ones. This approach is similar to the CORBA IDL to C++ mapping. The whole framework bears a lot of similarities to the OSIMIS GMS, which was input to the relevant standardisation work.

3.6.3 The Overall Architecture

The key aspect behind an object-oriented agent infrastructure is to provide an environment which supports the development and deployment of managed objects that form an agent "cluster". Management application developers should be able to concentrate in the provision of the associated behaviour, without being concerned about the CMIS access details and underlying protocol aspects. This implies that the APIs for such an infrastructure should be defined at the managed object level, in a symmetrical fashion to the Shadow MIB infrastructure. The managed objects in this case are the *master* objects, implementing the information aspects dictated by a TMN Q_3 or X interface.

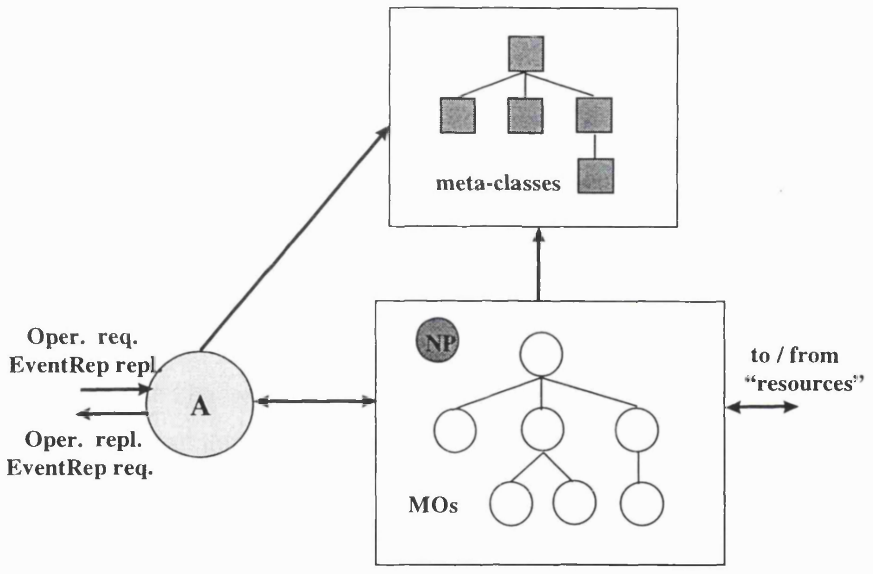
Given the object-oriented nature of GDMO as information specification language, it is most natural to map the abstract managed objects onto concrete C++ object instances. The exact mapping of the various GDMO features to a language like C++ is a difficult task because of the object model differences; this mapping will be discussed in detail in the next section.

A managed object should be able to respond to the CMIS get, set, action and delete primitives, should know its name, should keep handles to its superior and first level subordinate objects in the MIT and should be able to evaluate filters according to the Management Information Model (MIM) [X720]. We have thus identified a core component of an agent application, the *management information tree*.

An object instance contains information pertinent to that instance i.e. its attributes and other instance variables that relate to its behaviour and state. Management requests passed to it need to be validated e.g. verify that the relevant attributes or action type are supported by the instance, convert the raw *any* values to specific types according to the associated ASN.1 syntaxes, etc. Keeping this *meta-information* on a per instance basis means that it would be "n-plicated" for *n* instances of the same class. Since the information is related to the class rather than the instance, it should be kept in objects which model the managed object classes. We have thus identified another agent component, the *meta-class* objects. Given the fact that classes are related through inheritance, meta-class objects should be organised in a hierarchy that models the GDMO inheritance tree. These class objects can also serve as "factories" for the relevant MO instances.

Access to managed object instances should be provided through a component that supports the CMIS/P functionality i.e. performs CMIS/P PDU processing, resolves distinguished names to MO handles, evaluates scoping and filtering, provides access control, takes care of atomic transactions and forwards replies and event reports to manager applications. We will call this component the CMIS or MIB agent. This can be modelled by a single C++ object instance that encapsulates the Q_3 protocol stack. The CMIS or MIBAgent is in fact similar to the RMIBAgent in manager applications. The difference is there is only a single instance per agent application.

A final consideration is event reporting and logging. EFDs and logs will be obviously part of the MIT but the notification processing function, as specified in [X734][X735], should be a separate object which receives notifications and evaluates them. It may subsequently instruct an EFD to forward the event report through the MIB agent or instruct a log to create a new log record. The Notification Processing (NP) processing function or object can be thought as part of the MIT component. It is "invisible" though by the MIBAgent which accesses only managed objects.



Oper.: get, set, action, create, delete, cancel-get
NP: Notification Processing
MO: Managed Object

Figure 3-13 Object-Oriented OSI-SM Agent Decomposition

The initial decomposition of the agent application is shown in Figure 3-13. A management request always arrives to the agent object. In the case of a m-create, the agent resolves the superior and reference names to MO handles, locates the relevant meta-class object and requests for the object instance to be created. In the case of any other request, the agent resolves the base object name to a MO handle, evaluates scoping and filtering, checks the access control rights, performs the operations and returns the replies. A notification emitted by a MO is passed to the notification processing function which may instruct an EFD to forward it through the agent. If the notification is confirmed, the agent will pass eventually the confirmation back to the EFD.

An important design decision concerns the MIT representation of a MO and the actual MO itself. In the architecture presented above, these are mapped to the same C++ object instance. Another approach would be to keep them separate so that the managed objects could be in different operating system processes, using a proprietary distribution mechanism. In this case, the "core agent", i.e. the agent object, MIT representation and class dictionary could be in one operating system process while the managed objects and associated meta-class objects could be distributed in different processes. This approach has been adopted by a number of commercial products, with distribution supported through proprietary lightweight object brokers. OSIMIS and a number of other products follow a more "monolithic" approach, in which an agent application maps always to a single operating system process.

There are advantages and disadvantages with both approaches. The obvious advantages of the distributed approach is scalability and modularity: very large agent applications can be physically distributed. In addition, new functionality may be added to an agent in terms of new or modified classes by simply "detaching" and "attaching" new processes without having to shutdown the agent. This is an important requirement in telecommunications environments: a "cold start" of the management part of a network element may be undesirable. The key drawback of this approach is it complicates the agent architecture, necessitates the design of an object broker and results in slower response times. This approach goes to some extent beyond the OSI-SM model and introduces an "intra-application" distributed object framework, with Q_3/X interfaces retained for "inter-application" interoperability.

The author considered distributing the OSIMIS agent infrastructure but this meant essentially designing and implementing an object broker in addition to OSI-SM. The whole concept behind OSIMIS was to prove the feasibility, implementability, economy and performance of OSI-SM and not to provide a more general framework for distributed objects. Now that CORBA platforms exist, it would be relatively easy to distribute an agent application and this is a direction that a number of TMN platform vendors are taking. On the other hand, if CORBA is to be used within an application, it might be also used across applications and replace OSI-SM completely. This possibility is examined in detail in Chapter 4.

3.6.4 A GDMO to C++ Mapping for Managed Objects

In section 3.5.4 we discussed the mapping of GDMO to *shadow* managed objects in manager applications. Here we consider the mapping to *master* managed objects in agent applications.

A managed object is part of the MIT, so it should keep handles to the superior and first level subordinate objects. In addition, it should know its relative name i.e. the name of its distinguished attribute and value. In distributed agent environments, the MIT is a tree of "minimal" objects that contain only this information together with a reference to the actual managed object. This reference is according to a broker mechanism and can be used to invoke operations to the object. In centralised agent environments, the managed object is part of the MIT and responds directly to management operations. OSIMIS takes the approach of a centralised agent. The functionality described above is provided by the *MO* class, which is the root of the C++ inheritance hierarchy for managed objects.

```
class MO
{
    // . . .

public:
    MO*         Resolve    (DName* name);
    int         CheckClass (OIDentifier* cmisClass,
                            Bool& allomorphic);

    MO*[]       Scope (CMISScope* cmisScope);
    Bool        Filter (CMISFilter* cmisFilter);

    CMISErrors  Get (OIDentifier* objClass,
                     int nattrs, OIDentifier*[] attrIds,
                     int& nres, CMISGetAttr[] resAttrs,
                     AVA*& errInfo);
    CMISErrors  Set (Bool confirmed, OIDentifier* objClass,
                     int nattrs, CMISSetAttr[] attrs,
                     int& nres, CMISSetAttr[] resAttrs,
                     AVA*& errInfo);
    CMISErrors  Action (Bool confirmed, OIDentifier* objClass,
                     CMISParam* actionInfo, CMISParam *actionResult,
                     AVA*& errInfo);
    CMISErrors  Delete (AVA*& errInfo);

    // . . .
};
```

Code 3-11 The MIB Agent to Managed Object Interface

Since an object knows its relative name and has access to its first level subordinates, it can provide a *find* or *resolve* method which maps a distinguished name to an object handle through a recursive descent algorithm. In the same fashion, it is easy to provide a method that evaluates the CMIS scope parameter. The CMIS filter can also be evaluated if this class has access to all the attributes of derived classes. The key methods of the MO class which provide an interface to the MIB agent object are shown in the Code 3-11 caption.

The GDMO *top* class [X721] maps to a top C++ class which derives from MO. All subsequent GDMO derived classes should derive from top and mirror the GDMO inheritance in C++. There exist two key issues with inheritance in GDMO and inheritance in C++:

a) how GDMO multiple inheritance will be handled in C++; and

b) how behavioural aspects of GDMO managed object classes will be implemented in the corresponding C++ classes.

The immediate answer to the first question is to use C++ multiple inheritance to model the GDMO multiple inheritance. The problem with this approach is that it will not work with languages that do not support multiple inheritance such as Smalltalk [Gold83] and Java [Sun96]. Even in C++, it is necessary to eliminate duplicate attributes which should only appear once in a MO instance . In addition, method resolution is required when the same method appears more than once in parallel branches of the inheritance tree. The OSIMIS approach is to collapse the GDMO multiple inheritance into single inheritance and implement this in the standard C++ fashion. This is also an approach followed by many commercial products and there exist various algorithms for collapsing multiple to single inheritance. This can be done either automatically, by a GDMO compiler with a built-in algorithm, semi-automatically by the human user guiding the compiler, or manually. OSIMIS uses the third approach i.e. the user has to collapse manually GDMO multiple to single inheritance.

Adding behaviour to a class can be handled either by adding methods to the relevant C++ class or by "class doubling", as in [Doss93], [Chat97] and [CORBA]. In the latter case, behaviour is added through a derived class e.g. *discrImpl* for the *discr* class. The problem with this approach is that multiple inheritance becomes necessary for classes in levels deeper than two in the inheritance hierarchy. For example, the *eventForwDiscrImpl* class should inherit both from the *eventForwDiscr* class and from the *discrImpl* class. The OSIMIS approach is to include methods with behaviour rather than use class-doubling. This is done by files with the suffixes *.inc.h* and *.inc.cc* which should be present while compiling the GDMO specifications. This approach works with any object-oriented programming language that supports only single inheritance.

The next important issue concerns packages. An initial thought would be to map those to separate C++ classes, so that they become reusable entities. The problem is that while conditional packages are re-usable when considered as units of specification, their behaviour is in most cases dependent on the class in which they are contained. As such, the case for re-usability is weak while the complexity of a MO instance is increased. OSIMIS models the functionality of mandatory and conditional packages through the C++ class that models the GDMO class. This means that packages do *not* map to separate C++ classes. An instance keeps state information regarding which conditional packages are "active" so that it can respond correctly to management requests.

Attributes map to C++ classes that model the corresponding ASN.1 types. When an object instance is created, the attributes of the mandatory and active conditional packages form an array. Such arrays exist for all the classes of the inheritance branch, e.g. for *top*, *discr* and *eventForwDiscr*, and their handles are passed to the MO part at construction time. Since the latter has access to all the attributes of the instance, it can evaluate CMIS filters and respond to "get all" operations.

Polymorphic get, set, action, delete and create methods are defined by the MO class and may be redefined by derived classes to add behaviour. The get and set methods operate on a per attribute basis and are called for every attribute in the m-get and m-set request. They inform the object instance to either "refresh" those attributes or to do something to the associated resource as a result of a set request. The actual get and set operations to the attributes are performed transparently by the MO class. The action method instructs the behavioural part of an object instance to perform the action to the associated resource. The delete and create methods trigger associated behaviour, if any. Finally, the "trigger notification" method is called by the behavioural part of an MO to emit a notification.

```
class MO
{
    // . . .

protected:
    virtual int get (int attrId, int classId, AVA*& errorInfo,
                     Bool checkOnly = False, int asyncInvokeId = -1);
    virtual int set (int attrId, int classId, ModifyOp mode,
                     Attr* setVal, Attr*& resVal, AVA*& errorInfo,
                     Bool checkOnly = False, int asyncInvokeId = -1);
    virtual int action (int attrId, int classId,
                     Attr* info, Attr*& reply, AVA*& errorInfo,
                     Bool checkOnly = False, int asyncInvokeId = -1);
    // . . .
public:
    virtual int ddelete (DeleteType dtype, AVA*& errInfo,
                         Bool checkOnly = False,
                         int asyncInvokeId = -1);
    virtual int create  (CreateType ctype, AVA*& errInfo,
                         int asyncInvokeId = -1);


    int         triggerNotification (int notifId, int classId,
                                     Attr* notificationInfo);
    // . . .
};
```

Code 3-12 The Polymorphic Managed Object Methods

The polymorphic MO methods are shown in the Code 3-12 caption. A dynamic, weakly-typed approach has been followed, in a similar fashion to the RMIB model. If the GMS was to be redesigned now, the author might have followed a static, strongly-typed approach, similar to the NMF GDMO/C++ [Chat97] and CORBA [CORBA]. On the other hand, a static approach is much more complex and results in the generation of more code. The reader is reminded here that the advantage of the strongly-typed approach is compile-time as opposed to run-time checking. An important aspect of the polymorphic API is that these methods can be asynchronous as well as synchronous. The asynchronous option is typically used in single-threaded environments when the relevant method involves remote access of a subordinate information model.

Finally, aspects pertinent to a managed object class are modelled by a meta-class object. The general meta-class is called *MOClassInfo* and keeps information about the class, name bindings, packages, attributes, actions, notifications and the respective syntaxes and OIDs. The MO part of an object instance is told about the actual class by its leaf-most derived class and keeps a handle to the relevant class object. This means it has access to all the meta-class information so that it validate and handle a particular management request. Figure 3-14 shows the layout of a uxObj instance and its access to meta-class object instances.

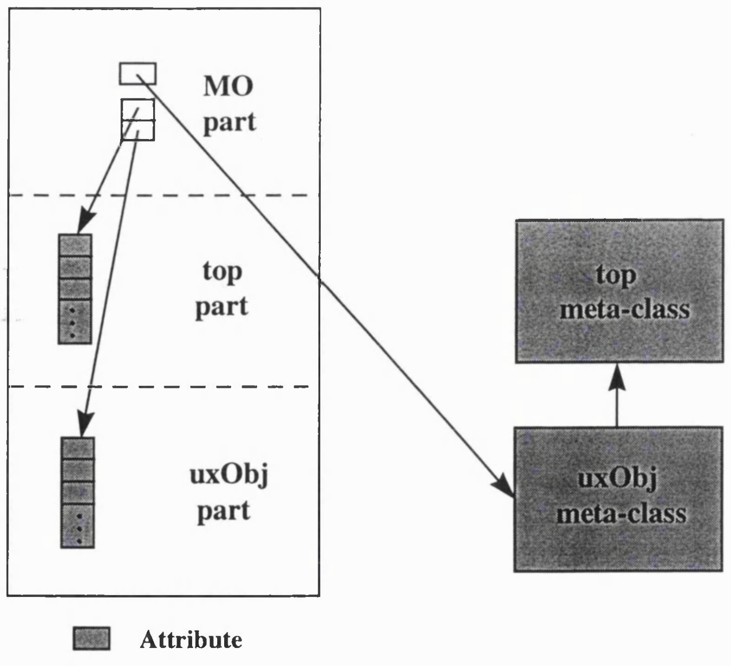


Figure 3-14 The Internal Layout of a uxObj Instance and Associated Meta-Class Objects

In summary, the rules for mapping GDMO to C++ presented above are the following:

- Managed object classes map to C++ classes. In the case of GDMO multiple inheritance, this is collapsed first to GDMO single inheritance and subsequently mapped to C++ single inheritance. The root of the C++ inheritance hierarchy is the MO class.
- Packages are implicitly present in managed object instances through their attributes, actions and notifications. The attributes of conditional packages are only instantiated if the package is present. An object instance keeps state information about the active conditional packages so that it can respond correctly to requests.
- Attributes map to C++ instances that model the relevant ASN.1 type. Every class keeps an array of its attributes and passes its handle to the MO part, which has access to all the attributes of that object instance. The get and set polymorphic MO methods are called to refresh an attribute value or to perform an operation to the associated resource.
- Actions map to the polymorphic action method which is called as a result of an action request. The action information and reply map to C++ instances that model the associated ASN.1 type.
- Notifications are supported by the triggerNotification method of the MO class which is called when a notification is emitted. The notification information maps to a C++ instance that models the associated ASN.1 type.
- Parameters to attributes and actions indicate MOC-specific errors and are mapped onto C++ AVA instances in the relevant polymorphic methods.
- Name bindings map to special C++ instances kept by the meta-class object. Every object knows its name and the active name binding through the nameBinding attribute of the top class.
- Finally, any other information pertinent to the class is mapped to instance variables of the meta-class object.

This approach has been validated by the implementation of the OSIMIS GMS which has been used to realise various applications in agent roles as described in Appendix A. Most of the abstractions presented above have been adopted by a number of commercial products. The presented APIs hide the underlying protocol aspects as much as possible and leave an implementor to concentrate in the behaviour of the managed objects, which is provided by the redefinition of the polymorphic methods.

As an example, the implementation of the uxObj class presented in Appendix B involves the following. The *get* method should be redefined to refresh the *sysTime* and *nUsers* attributes when these are requested by name or when all the attributes are requested; and the *action* method should be redefined to respond to the *echo* action. The set method does not need to be redefined for the *wiseSaying* attribute since there is no associated resource. The agent and the MO part perform transparently all the checking, decoding, encoding and behaviour triggering. The *objectCreation*, *objectDeletion* and *attributeValueChange* notifications are also emitted transparently, without any additional code by the implementor. The GDMO compiler produces the object skeleton in *uxObj.h* and *uxObj.cc* files while the behaviour is included through *uxObj.inc.h* and *uxObj.inc.cc* files supplied by the implementor. The uxObj implementation involves only a few lines of code.

3.6.5 The Relationship of Managed Objects and Associated Resources

A managed object always represents a real resource. This may be a fine-grain resource at the lowest level of the TMN hierarchy, i.e. in network elements, or a more abstract resource in higher TMN layers. Network element resources can be thought as tightly-coupled with the associated managed objects since they are co-located in the same network node e.g. an ATM switch. In this case, access time between objects and resources is typically small. Resources in higher TMN layers can be also tightly coupled with the associated managed object, e.g. a customer record in the service management layer. Most typically though, a resource in a higher TMN layer is loosely coupled with the managed object in the sense that it maps onto lower layer resources in a recursive manner. In this case, an operation to the associated managed object may result in operations to subordinate managed objects as described in Chapter 2.

An important issue is how the attributes of the managed object will present a consistent and up-to-date view of the resource in the case of m-get operations. There are three major approaches for maintaining the MO and associated resource consistency:

a) "*access upon request*" - the resource is accessed only as a result of the m-get request;

b) "*cache ahead periodically*" - the resource is accessed periodically and the relevant attributes are refreshed; and

c) "*update through events*" - in this case the resource sends changes to the managed object asynchronously and this updates its attributes.

The first approach has the advantage of reduced management traffic in the case of loosely coupled resources but at the cost of increased response time. In the case of tightly-coupled resources within network elements, it is the most sensible approach. The second approach has the disadvantages of increased network traffic and potentially reduced information timeliness. Its only advantage is fast response time since the managed object is able to respond immediately to the m-get request. The third approach generates less traffic than the second one and retains the advantage of fast response. On the other hand, it incurs more management traffic than the first one. When using b) and c) above, the polymorphic get method does not need to be redefined since the attribute values will be up-to-date upon the reception of a get request.

There can be variations and combinations of those approaches. OSIMIS supports all three methods of operation: a) by redefining the polymorphic get method, b) through support for periodic polling (it will be described in section 3.7.2) and c) through events from subordinate systems.

3.6.6 Realisation of the "Difficult" Agent Aspects

In this section we look at those features of OSI-SM which are considered difficult to implement and explain how they are supported by the underlying agent infrastructure.

The MIT is a n-ary tree which can be represented internally as a binary tree. The manipulation of binary trees has been addressed in detail in the literature. The author used data structures and algorithms described in [Aho83]. An issue related to the MIT representation is CMIS scoping. This is almost trivial to provide: a recursive method is used which performs a pre-order search down to the required level and adds object handles to an array.

Another aspect that was initially considered difficult to implement is filtering. Attributes are mapped to C++ Attr instances which can evaluate filter assertions as explained in section 3.4. A CMIS filter is a tree data structure in which all the leaves are filter items (see Appendix D). A recursive algorithm can be used to scan the filter expression and evaluate the filter items linked by boolean operators. The attribute name in the filter item is first verified through the meta-class information and the relevant assertion is then evaluated. The filter method is supported by the MO class which has access to the meta-class information and the attributes of the object instance. Before filtering is applied, the object instance is requested to "refresh" the relevant attributes.

Resolution of distinguished names is also supported by the MO class. Since every MO knows its relative name and keeps handles to superior and subordinate objects in the MIT, a recursive method can perform a breadth-first search and compare relative names. This means that the associated computing cost is generally a linear function of the breadth and depth of the MIT for a particular object. Another approach which could result in a flat response time is a hash-table approach. In this case the choice of the hash algorithm becomes crucial in order to result in a relatively even distribution and avoid collisions which cause inefficiencies.

One of the most difficult OSI-SM aspects is atomicity which may be requested through the CMIS synchronisation parameter. OSIMIS implements atomicity through an internal two-phase commit approach. Initially all the selected managed objects are asked by the agent if they can perform the requested operation. This is done by calling the relevant polymorphic method with the *checkOnly* flag set to *True* (see Code 3-12). If at least one object does not accept, the operation is rejected. Otherwise, the objects are requested to commit the operation. This approach provides a rudimentary atomicity facility. Distributed transaction processing [DTP] may be used in conjunction with CMIS to "bracket" a number of operations to different agents in an atomic transaction.

Another difficult aspect related to the realisation of the GDMO model is allomorphism. This is supported in the following fashion. The MO part initially checks the asserted class and makes sure that the attributes or specific action are supported by it (or by its parent classes). The behavioural code of a leaf most class subsequently checks the class asserted in the operation. If this is one of the parent classes, it calls the equivalent method of the parent class. In this way, the behaviour of the right class is always invoked even if an attribute or action has been redefined in a derived class. This implies that support for allomorphism should be explicitly provided by the behavioural parts of managed objects.

Finally, managed objects need to be persistent so that they can survive re-starts of the agent. Another reason for object persistence is that it may be impossible to hold all the managed objects in core memory in the case of agents with a very large amount of objects. Object persistence can be provided by storing the state of an object through its attribute values in secondary storage. The fact that attributes in OSIMIS have a well-defined string representation can be used to serialise a MO and "save" it on disk. A database with random access is necessary to "revive" a particular MO. The GNU version of the UNIX *dbm* database was used in OSIMIS for object persistency.

3.6.7 Systems Management Functions

The OSI Systems Management Functions (SMFs) [SMF] have been introduced in detail in Chapter 2. In this section we consider briefly their agent realisation aspects.

The most important SMFs are event reporting [X734] and logging [X735]. Their rationale and model is explained in [Laba91a]. When a managed object emits a notification, this is passed to the notification processing (NP) function. The latter could be modelled by a separate object as it was depicted in Figure 3-13 but in the case of OSIMIS it is part of the MO class. A notification is triggered by the behavioural part of a managed object through the *triggerNotification* MO method. A "potential event report" or "potential log record" is subsequently created. This. is a specific log record object according to the notification type e.g. an *attributeValueChangeRecord.* This object is not part of the MIT but serves solely the purpose of evaluating the EFD and log filters. The NP runs subsequently through the EFDs and logs and evaluates their filter on the potential event report. If the filter evaluates to true, it instructs the relevant EFD to send the event report or instructs the relevant log to create a copy of the log record. The event report is sent through the agent. The EFD may need to retransmit confirmed event reports until a confirmation is received through the agent.

The power of the OSI-SM event reporting model stems mainly from the fact that EFDs and logs may contain sophisticated filters which can be applied to the potential event report objects. This allows assertions on the class and name of the emitting object, on the event type and time and on information specific to the event e.g. the attribute name, its previous value and its new value for an *attributeValueChange* event.

Other important SMFs are object [X730] and state [X731] management. The object lifecycle notifications, i.e. *objectCreation*, *objectDeletion*, are automatically supported when an object is created and deleted. In addition, the *attributeValueChange* and *stateChange* notifications are also supported automatically when the attribute changes as a result of a CMIS m-set request. The relationship management SMF [X732] specifies relationships through "pointer" attributes that contain the distinguished name of another object. These can be resolved to the pointed object handles through the find or resolve MO method.

The only other SMF which has an impact on the agent infrastructure is access control [X741]. The agent object will contain an Access Decision Function (ADF) object which will check access control rights and will grant or deny access for a management operation. The MIB agent is seeded with hooks which will invoke the ADF functionality and check access rights.

The rest of the SMFs consist simply of support managed objects which are implemented and linked with agent infrastructures so that they can be instantiated by manager applications. OSIMIS supports the SMFs mentioned above and also the metric monitoring objects [X739], the summarisation objects [X739] and a combination and extension of the previous two known as the intelligent monitoring objects [Pav96c].

3.6.8 Summary

In this major section we looked at issues related to the realisation of object-oriented agent infrastructures. The target was to provide an object-oriented decomposition of an agent infrastructure, propose a concrete mapping of GDMO to object-oriented programming languages and provide an environment for the realisation of managed objects which shields implementors from the underlying access service and protocol complexity.

We proposed an agent architecture known as the Generic Managed System which separates service and protocol processing from the managed objects through an MIB agent object. We also proposed a GDMO mapping to C++ which uses only single inheritance and can be provided in languages such as Smalltalk and Java. The managed object class specifications are compiled through a GDMO/ASN.1 compiler which produces stub MO classes. These can be augmented with behaviour by redefining the polymorphic methods of the generic MO class. The attributes, action and notification information are modelled though C++ instances according to the O-O ASN.1 principles presented in section 3.4. The resulting environment shields implementors from protocol details and enables them to concentrate in the object behaviour and associated application intelligence. The whole approach is very similar to CORBA but was designed, specified and implemented much before the CORBA IDL to C++ mapping.

We also discussed issues related to the maintenance of consistency between managed object attributes and associated resources. Three different schemes were identified which have different properties in terms of incurred management traffic and information timeliness. We finally discussed the "difficult" implementation aspects of OSI-SM and explained that they are in fact easy to provide through object-oriented design principles. These included the MIT representation, scoping, filtering, name resolution, atomicity, allomorpism and persistence. We also explained how event reporting, logging and the rest of the OSI SMFs can be supported.

In summary, this section demonstrated the feasibility and implementability of the agent aspects of the OSI-SM model. The author designed and implemented the majority of the agent infrastructure. He would like though to thank other members of the UCL team for their contributions and in particular: G. Knight for various discussions and direction and also for conceiving and implementing the uxObj example; J. Cowan for implementing the GDMO compiler in an ingenious fashion so that its back-end can be easily customised; S. Bhatti for designing and implementing the log control SMF and the managed object persistency; and K. McCarthy for trying to "break" the agent infrastructure in every conceivable fashion while implementing the generic CMIS/P to SNMP gateway.

3.7 Issues On Synchronous vs. Asynchronous Remote Execution Models

In the previous sections of this chapter we have demonstrated how to map the OSI-SM/TMN model onto O-O programming environments in the form of a distributed management platform. The realisation model for both the agent and manager infrastructure included both synchronous and asynchronous API facilities. In this section, we investigate the relevant issues behind the use of synchronous and asynchronous options. We also present the solution adopted in OSIMIS for co-ordinating the activity of a management application.

3.7.1 Remote Procedure Call and Message Passing Paradigms

The synchronous remote execution model was first presented in [Birr84] through the notion of Remote Procedure Calls (RPCs). A remote operation is modelled through a procedure in the calling entity, with semantics similar to those of local procedures, which takes as input and output arguments those of the remote operation. When this procedure is called, it triggers the actual remote operation to another entity, possibly across the network, and returns with the result or error. While a remote procedure call looks exactly like a local procedure call from a programmatic point of view, it has very different performance characteristics. A message needs to descend and ascend the local and remote protocol stack, including the relevant encoding and decoding overhead. In addition, the end-to-end network latency for the request and response adds to the overall delay. In general, the difference in performance between local and remote method calls is at least two orders of magnitude (10^2).

The asynchronous remote execution model predated the RPC model and is often referred to as "message passing". A remote operation is modelled through two different operations in programmatic terms: a "send" procedure, which takes as arguments only the input parameters for the operation; and a "receive" procedure", either in a callback [Clark85] or in a message queue fashion. In the case of the callback or "push" model, the "send" procedure also passes as an argument (a pointer to) the "receive" procedure, which is eventually invoked by the infrastructure with arguments the output parameters for the operation. In the case of the event queue or "pull" model, the "receive" procedure needs to be invoked by the caller. The receive procedure is typically generic, i.e. the same for all the different send procedures, and fills-in a data structure which is the "union" of all the possible output parameters for the remote operations.

The performance cost of calling the send or receive procedures is exactly that of descending or ascending the protocol stack. This means that the relevant cost is only a fraction of the total cost of a remote procedure call. Exact figures of this difference in the case of CMIS/P will be presented in the next section. According to the models described above, the MSAP API is an asynchronous one using the event queue model while the RMIB/SMIB and GMS APIs are both synchronous and asynchronous, using the callback model.

The main advantage of the synchronous execution model is that it is natural to the programmer: a remote method call appears exactly like a local method call, so the logical flow of a program that uses remote operations is not different to a program that invokes no remote operations at all. The RMIB and SMIB examples in the Code 3-6 and Code 3-7 captions used the relevant API in a synchronous fashion. In comparison, the asynchronous execution model appears less natural for the programmer: every time a remote method is invoked, the logical flow of the program is "interrupted" i.e. the method that contains the "send" method invocation typically keeps some state and returns. If, say, the callback model is used for the result, the logical flow of the program will continue again when the callback method is invoked with the result. In summary, the asynchronous execution model necessitates to keep some state information while it also splits the logical flow of control in two parts: one before the remote method is called and one after the callback method is called with the result, the two parts being in different program methods.

While from the above discussion it appears that a synchronous execution model is more natural, and thus desirable, it also has a relevant limitation which requires special support. Synchronous method calls block the performing application until they return (by application in this case we mean a single operating system process). Since the cost of remote method call is much higher than that of a local method call, the overall performance of the application will degrade since it will stay idle for relatively long periods of time, while there may something else that needs attention. For example, a TMN OS may be performing a number of sequential synchronous operations to subordinate OSs while a peer or superior OS has sent a request to it which is being "queued up".

In the above discussion we have assumed a single-threaded execution paradigm i.e. the application is a "heavyweight" operating system process that does not contain "lightweight" processes or threads internally. The answer thus to the above performance problem is to use multi-threading. The latter though introduces the need for *concurrency* control: different threads inside the same program may try to access the same data, so some form of locking is required. This in fact requires state information while it also introduces the possibility of deadlocks. In fact, it is difficult to harness the power of multi-threading in complex programs such as a TMN OSs, where many things make take place in parallel, and make sure they will never deadlock.

An additional issue with multi-threading is the fact it is not yet supported by every operating system in a native fashion, i.e. through the operating system kernel. For example, it is supported in the Sun Solaris version of UNIX and the MS Windows NT but it is not yet supported in the increasingly popular Linux version of UNIX. In fact, "native" threads have only been available during the last 2-3 years. In the past there had been user-space thread packages which did not have proper support for system facilities. They made programs difficult to debug because debuggers were "unaware" of them. In addition, they added their own performance overhead. The author had some "interesting" experiences with the ANSA [ANSA89a] user-space threads package before he decided to switch into single-threaded execution mode to solve those problems. It should be noted though that nowadays multi-threading has become much easier to use with native operating system support and languages like Java [Sun96] which provide built-in support.

When OSIMIS was first designed (circa 1989-90), it was necessary to support asynchronous APIs for applications that were conscious about performance, in addition to synchronous ones. It is interesting to observe that the NMF TMN/C++ family of APIs, which represents a serious industrial approach towards TMN platform APIs, provides *both* synchronous and asynchronous APIs, in a similar fashion to OSIMIS. It is also interesting the fact that OMG has in its plans a specification an asynchronous API facility for CORBA version 3. The author has always expressed the necessity for asynchronous APIs, in addition to synchronous ones, and this seems to be in line with the current industry approach to distributed systems. Finally, [Crow93] points out problems of the synchronous remote execution paradigm when used without multi-threading.

OSIMIS was designed to operate initially in a single-threaded execution mode, leaving the introduction of multi-threading for a later stage. In fact, the latter never happened because the author got too busy with other things, but at least two commercial TMN platform vendors whose products are based on OSIMIS have introduced multi-threading. Given the fact that OSIMIS is single-threaded, asynchronous remote method execution is necessary for increased performance in complex applications. Asynchronicity requires state management, so it is necessary to support the application programmer through relevant platform facilities. As such, an object-oriented application *co-ordination* mechanism was designed and developed. This makes possible to organise a complex program that receives external input and generates output in a single-threaded fashion. The principles of this important mechanism are described in the rest of this section.

3.7.2 The OSIMIS Coordination Mechanism

The idea of a co-ordination mechanism with a central facility in each application that becomes the focal point of all external input was contributed by G. Knight of UCL in late 1989 i.e. in the very early days of OSIMIS. The author researched into the relevant issues and designed and developed this mechanism in an object-oriented fashion, using inheritance and polymorphism. A particular objective in this design was that it should be able to coexist with similar mechanisms of other systems. The only other system in mind at the time was the X Windows system, which was going to be used for TMN WS-OSs. The relevant design has been general enough though, so it has been possible to extend the mechanism later and add support for the Tk widget set [Oust94] and more recently for the Orbix CORBA platform.

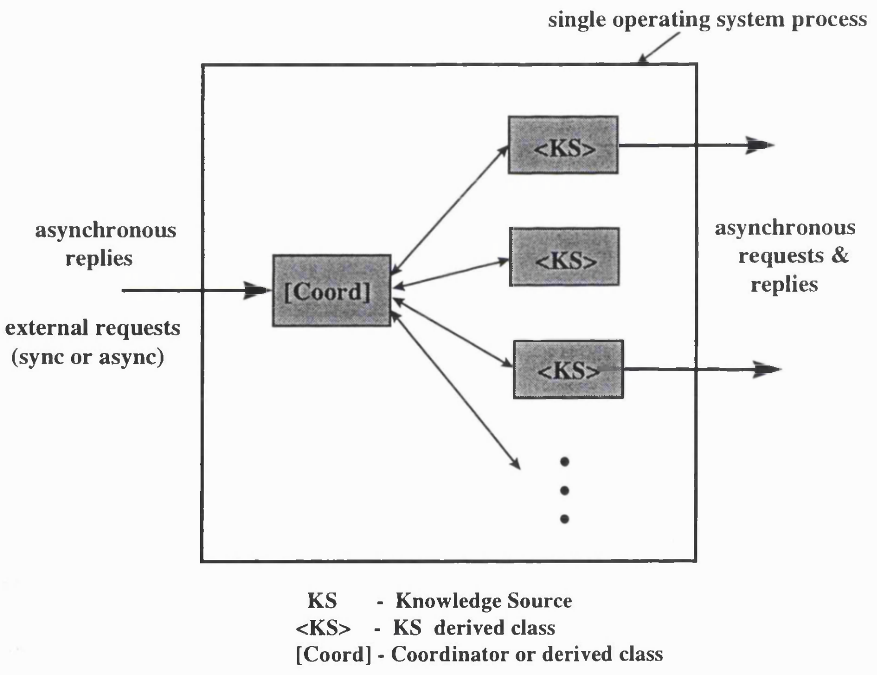

single operating system process
asynchronous replies
external requests (sync or async)
[Coord]
<KS>
<KS>
<KS>
asynchronous requests & replies
KS - Knowledge Source
<KS> - KS derived class
[Coord] - Coordinator or derived class

Figure 3-15 The OSIMIS Co-ordination Model

The OSIMIS co-ordination mechanism allows to organise a management application which is realised as a single operating system process in a fully event-driven fashion. All external input is serialised and taken to completion on a "first-come-first-served" basis. In every application, there exists a single instance of class *Coodinator* or of a derived class. This implements the central listening loop of the application, in fact the last line of the main program of any OSIMIS application is always "`coordinator.listen();`". Any other object in the system that expects external input should derive from the abstract class *KS* which stands for "Knowledge Source", its name having the roots in Artificial Intelligence (AI) Blackboard Systems. KSs register "communication endpoints" with the Coordinator and are informed when there is relevant input. In addition, the Coordinator handles timer alarms and wakes-up periodically KSs that have requested this facility. This is because most operating systems do not "stack" timer alarms, so a central facility for taking care of them is necessary.

This model is depicted in Figure 3-15. The various KSs in the system should perform only asynchronous operations, with the replies passed back to them through the Coordinator. Every time there is external input or a timer alarm expires, the Coordinator informs the relevant KS who starts some activity. During the period of this activity, any other external input or timer event will have to be queued up. This means it is up to the objects involved in a particular sequential thread to relinquish control i.e. this is *not* pre-emptive round-robin scheduling. As a consequence, time-consuming activities should be ideally delegated to other processes so that the system remains responsive. For example, if as a result of a number of alarms, a TMN OS needs to evaluate a large rule base which is expected to take time, the rule-based system should be ideally implemented in another process that communicates with the OS through some Inter-Process Communication (IPC) mechanism, e.g. shared memory. Of course, nothing prevents any part of the application from performing time consuming activities, invoking remote operations in a synchronous fashion etc. This attitude though is not recommended since it will affect the application's performance and responsiveness.

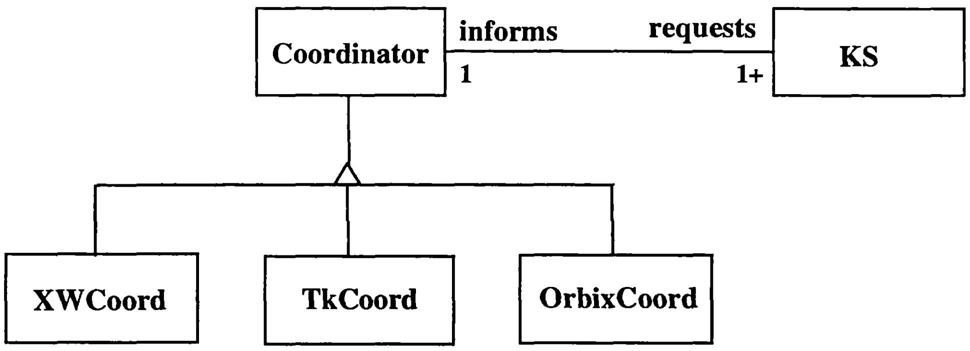


Figure 3-16 OMT Relationships of the Co-ordination Classes

In a TMN OS realised using OSIMIS, the only KSs that expect external input are the MIBAgent and the various RMIBAgents. There may be of course many other KS objects that require to be "awaken up" periodically. This mechanism has been tried and tested with complex TMN applications and has proved remarkably successful. In fact, this mechanism could also provide the basis for introducing multi-threading: the central listening loop will be realised in a central thread, while new threads will be spawned for every new external input or timer alarm. While this is trivial to introduce, the key issue becomes then concurrency control and this involves a lot of work. One commercial product which is based on OSIMIS has introduced multithreading exactly in this fashion. In this case, pre-emption is taken care by the operating system. The advantage is that spawned activities need not be "conscious" anymore about their duration.

Figure 3-16 shows an OMT diagram of the relevant classes. There exists always one instance of Coordinator or of any derived class in the application while there exist one or more KSs. These request to be informed of external input or to be informed at regular time intervals and the Coordinator subsequently informs them. The Coordinator derived classes allow OSIMIS applications to coexist with other systems. For example, OSIMIS-based TMN WS-OSs may use X Windows or the Tk widget set. In addition, OSIMIS applications may contain CORBA objects, as it will be explained in Chapter 4.

```
class KS {
protected:
    // request wake-ups, register commEndpoints with Coord

    int scheduleWakeUps (long period, char* token = NULL);
    int cancelWakeUps   (char* token = NULL);

    int startListen (int commEndpoint);
    int stopListen  (int commEndpoint);
    // . . .

public:
    // callbacks for wake-ups, external events, process shutdown

    virtual int wakeUp (char* token);
    virtual int readCommEndpoint (int commEndpoint);
    virtual int shutdown ();
};
```

Code 3-13 The O-O Coordination API - the KS Class

The Code 3-13 caption above shows the object-oriented co-ordination API realised by the KS class. This provides methods to register external communication endpoints and to request periodic "wake-ups". Derived classes should implement the behaviour of the callback methods. Note that the Coordinator class is not visible to KS-derived classes but is hidden behind this O-O API.

The author designed and implemented the OSIMIS co-ordination mechanism, which is the "heart" of OSIMIS-based applications, early in the design of OSIMIS. It does not involve a lot of software in terms of lines of software but it took some time to get the object-oriented design right. In addition, the implementation of the Coordinator class and the relevant derived classes contain very complicated and relatively low-level software. Through object-orientation though this complexity is encapsulated and reused. The OSIMIS co-ordination mechanism was quite unusual at the time and still stands out for its object-oriented design and extensibility. It is now very similar though to the event model adopted for GUI implementation in many systems.

3.8 Performance Analysis and Evaluation

Object-oriented technology is generally thought to be expensive in terms of both processing and memory overhead, in comparison to non object-oriented systems. In addition, OSI technology is also considered expensive and non-performant. Since the proposed TMN framework is realised using object-oriented technology over OSI protocols, it is interesting to evaluate the relevant performance overheads and attribute them to the various parts of the system. This is necessary in order to validate the proposed framework from a performance point of view. While we have already demonstrated its flexibility, simplicity, user-friendliness and support for rapid application development in the previous sections, the significance of those would be undermined if the proposed environment is expensive in terms of required resources, resulting in poor performance for the relevant applications.

In this section, we measure the performance of the proposed framework in terms of program size, response times and amount of communicated information for a number of carefully chosen "benchmark" operations. In the relevant measurements, we try to separate the protocol overheads from those of the OSI-SM application framework. This is particularly important since the latter may be mapped onto other environments and protocols, e.g. over OMG CORBA, as proposed in Chapter 4. The overhead of the protocol stack and the application framework are also analysed further. We measure the overhead of every layer of the protocol stack while we also measure particular aspects of the proposed application framework.

3.8.1 The Environment and Methodology Used in the Experiments

The environment for the experiments was the following. The applications ran on two different UNIX workstations, connected to a lightly-loaded Ethernet local area network. The round trip delay was 1.5-2 msecs measured with the UNIX *ping* program. The applications operated the full Q_3 stack over TCP/IP, using the RFC 1006 method to emulate TP0 [Q811] - the Q3 protocol stack was described in detail in section 3.3.1. TCP/IP ran in the UNIX kernel while RFC1006 and the rest of the upper layer stack were linked with the applications and, as such, ran in user space. The OSI upper layer stack, including ACSE, ROSE and DASE, and the QUIPU Directory Service Agent (DSA) [QUIPU] were provided by ISODE-8.0 [ISODE]. CMISE and the rest of the TMN application framework were provided by OSIMIS-4.0 [Pav95b], which is the environment resulted from the research work described in this thesis.

Two different pairs of UNIX workstations were used in the experiments:

a) a Sun SPARC 5 and a Sun SPARC 20, running Sun's Solaris 2.5 version of UNIX; in this environment, the Sun C++ and C compilers version 4.1 were used; and

b) a Dell Latitude XP 486-75 laptop PC and a Viglen Genie 486-100 desktop PC, running the Linux 1.12 version of UNIX; in this environment, the GNU C++ and C compilers version 2.6.3 were used.

The first pair represents typical UNIX workstations used in work environments while the second pair represents fairly old technology PC's. The reason for the second pair is that they are in fact the author's home desktop and laptop PC's. The second pair offered the possibility to conduct tests in a fairly isolated environment i.e. a home Ethernet LAN. All the experiments were conducted in such a way as to make sure that the programs were kept in core memory, so that no paging took place which could affect the relevant performance figures.

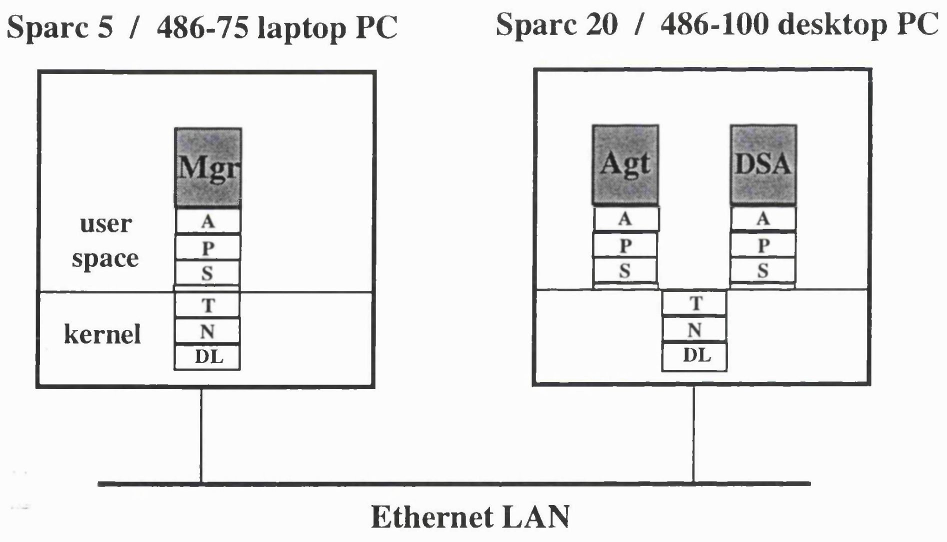


Figure 3-17 The Experiment Environment

All the experiments involved interactions between simple manager and agent applications. Since the computers in each of the two pairs were not the same, the agent application was run typically in the more powerful computer (i.e. the Sparc 20 and the 486-100 desktop). The experiments were also performed in the opposite direction i.e. with the agent running in the less powerful computer, without experiencing any significant differences in the pattern of the results. When a third application was involved, i.e. the DSA for location transparency, this was run at the same computer with the agent, so that the manager had to access it across the network. The environment for the experiments is depicted in Figure 3-17, showing the kernel / user space separation inside every workstation and the protocol stack supporting the application. Note that the transport layer runs partly in the kernel (the TCP part) and partly in user space (the TP0 emulation through RFC1006).

The methodology for the measurements was the following. Program size was measured using the UNIX *top* utility program, with measurements taken on both the Solaris and Linux systems. The resulting differences were only minor while the program sizes reported are those experienced on Solaris. Packet sizes were measured using the popular *tcpdump* program, developed at the Lawrence Berkeley Laboratory. The sizes reported reflect the TCP payload as introduced by higher layers i.e. TCP, IP and Ethernet headers are not included. This is desirable since the same upper layer stack can operate over different lower layers, as it was described in section 3.3.1. Response times were measured using the UNIX *gettimeofday* system call and comparing measurements taken on the same computer in order to avoid the need for synchronised clocks. Timestamps were taken at different points of a sequential execution thread within a program and were kept in core memory, to be printed out only after the last significant timestamp had been taken. This was in order to reduce the overhead and discrepancy introduced by the measurement mechanism itself. It is not though possible to minimise totally its impact, in fact the author observed behaviour similar to the principle of Heisenberg in atomic physics: when trying to observe minuscule things, the measurements themselves had a significant impact in the observed results. Every experiment was conducted a number of times. Unusual results were discarded while the mean was derived from the rest. The results were fairly uniform and the standard deviation small.

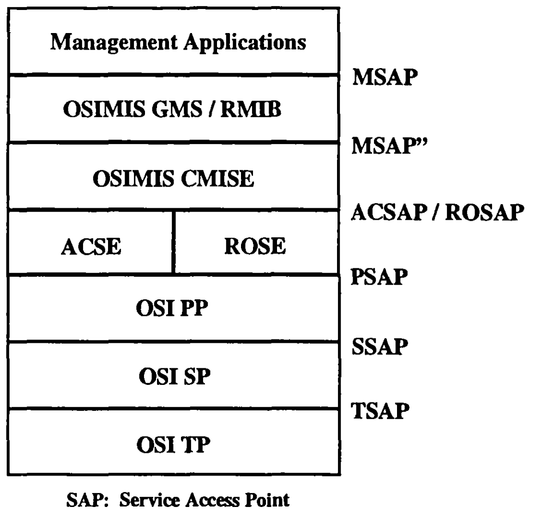


Figure 3-18 Service Access Points for the Performance Measurements

Special client and server applications were developed to echo data at every level of the protocol stack, including the TSAP, SSAP, PSAP, ACSAP/ROSAP and MSAP access points as shown in Figure 3-18. Two different notions of MSAP were used. The first was a "contrived" one, directly over the CMISE service without any agent and manager infrastructure, aiming to measure only the overhead of the OSIMIS CMISE layer over ACSE and ROSE; we will term this MSAP". The second is a "proper" MSAP, with the agent and managed objects in place through the GMS and the manager sending the request and receiving the response through the RMIB high-level API.

Response times in the TSAP and SSAP access points were measured using an "echo" operation with a byte string. Response times in PSAP were measured using an "echo" operation for an instance of the ASN.1 GraphicString type. Response times in ROSAP were measured by defining an *echo* remote operation with a GraphicString argument and result type. Finally, response times in MSAP" and MSAP were measured using the *echo* action of the uxObj class specified in Appendix C. The relevant CMIS request was the following:

```
m-action(objClass=uxObj, objName={uxObjId=null},
         actionType=echo, actionInfo=hello),
```

3.8.2 Program Size

We start first by examining the size of applications. Table 3-6 shows the size of client and server programs at the various SAPs identified above while Figure 3-19 depicts the same data in a graph form. A MSAP agent application with the object management [X730], event reporting [X734] and logging [X735] SMF capabilities and one instance of the system [X721] and uxObj classes amounts to 1340 Kb at run time. A MSAP manager implementing the echo action mentioned above amounts to 1060 Kb at runtime. Two similar but trivial MSAP" programs that contain no agent, manager or O-O ASN.1 infrastructure amount to 800 and 780 Kb or 59.7% and 73.5% of the overall size respectively as shown in Table 3-6. The equivalent ACSAP / ROSAP programs are 43.2% and 50.9% of the overall size respectively. This means that the overhead of CMISE together with the O-O TMN application framework amounts to 56.8% of the agent size and to 40.1% of the manager size. This can be explained as follows: CMISE is a fairly complex protocol compared to the rest of the underlying protocol stack. In fact, it is four times bigger than that of ACSE / ROSE according to those figures. The rest of the TMN application framework is pretty complex as well and it is implemented in C++ instead of C, which increases the relevant overhead. In summary, the overhead of the overall management infrastructure is roughly as much as that of ACSE/ROSE together with the rest of the underlying OSI protocol stack.

In TMN environments, OSs are hybrid manager-agent applications. If we link together the agent and manager described above so that echo actions can be performed in a peer-to-peer fashion, the run-time size becomes 1400 Kb. This is slightly bigger than the size of the plain agent, the difference (60 Kb) being the overhead of the RMIB infrastructure. This modest increase in size is easily explained since the two applications contain already a lot of common infrastructure i.e. the O-O ASN.1 support, the DMI syntaxes [X721] and the O-O co-ordination mechanism. In summary:

- the smallest OSIMIS-based TMN OS with object management, event reporting, logging and one "useful" managed object class with one relevant object instance amounts to 1400 Kb at run time.

Service Access Point	**Server Size (Kb - %)**	**Client Size (Kb - %)**
TSAP	390 - *29.1*	350 - *33*
SSAP	470 - *35*	420 - *39.6*
PSAP	530 - *39.5*	480 - *45.2*
ACSAP / ROSAP	580 - *43.2*	540 - *50.9*
MSAP"	800 - *59.7*	780 - *73.5*
MSAP	1340 - *100*	1060 - *100*

Table 3-6 Application Sizes at the Various Service Access Points

Another interesting aspect of the data in Table 3-6 is that the overhead of the RFC 1006 transport protocol appears to be around 30% of the overall client or server size. It is not the actual RFC 1006 code that causes this overhead, in fact this should be smaller than the session and presentation protocols which are around 80 Kb and 60 Kb respectively. This "initial fat" amounts to the overhead of the ISODE environment and, in particular, is due to the fact that the transport code has been written in a way to operate over a number of underlying protocols and environments. Even so, this overhead is pretty high (around 300 Kb) and could probably be reduced in a more carefully engineered protocol stack.

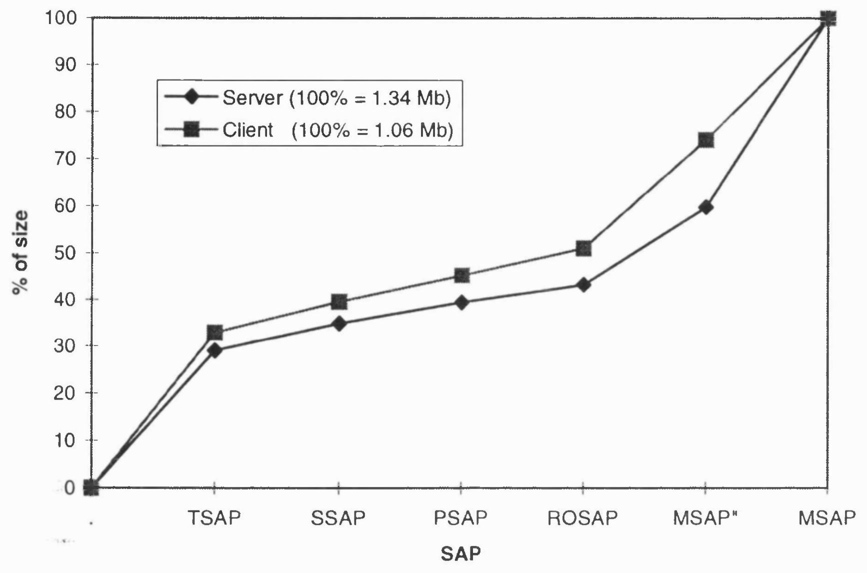


Figure 3-19 Application Sizes at the Various Service Access Points

We will finalise the discussion on the protocol stack overhead by examining the overhead of the CMOT instead of the full Q_3 stack. The CMOT protocol stack [Besa89] uses a Lightweight Presentation Protocol (LPP) [Rose88] that operates directly on top of the TCP. This is supposed to save mainly the session protocol overhead, given the fact that no sophisticated session layer services are needed for request-response protocols which are based on ROSE. Having rebuilt the ACSAP / ROSAP and MSAP programs over the LPP, it resulted in a reduction of about 150 Kb at runtime. This is only 10.7% of the size of the smallest TMN OS and it would be even smaller for OSs with real intelligence. As such, the CMOT memory savings are relatively insignificant.

Having examined the run-time memory overhead of the "base" management infrastructure, we will examine now the overhead of adding new classes, the overhead of object instances and the overhead of keeping management associations open. The overhead of linking in the uxObj class with the relevant behaviour amounts to 18 Kb at run time. The same overhead for the simpleStats class is 12 Kb. These two are example classes that were used for the purpose of demonstrating OSIMIS features. The performance monitoring classes *scanner*, *monitorMetric* and *movingAverageMeanMonitor* [X739] provide useful functionality for performance management and contain fairly complex behaviour. Linking those in amounts to another 76 Kb at run-time i.e. roughly 25 Kb for each of them. In summary, the minimal overhead of a new class with no behaviour seems to be about 12 Kb while the overhead for classes with significant behaviour can vary.

In order to evaluate the overhead of object instances, we implemented a simple manager that requests the creation of N instances of a particular object class. We used the uxObj class of Appendix C, with the values for the attributes depicted in the Code 3-14 caption. The data overhead of this particular instance is 420 bytes at runtime.

```
objectClass        2.37.1.1.3.50   (uxObj)
nameBinding        2.37.1.1.6.50   (uxObj-system)
uxObjId            <n>             (number in the 1-n range)
sysTime            19971223xxxxx   (time in UTC string form)
wiseSaying         "hello world"
nUsers             <m>             (small number e.g. 5)
```

Code 3-14 The uxObj Instance Used for the Memory Measurements

An agent containing 1 uxObj instance amounts to 1.34 Mb, 10^3 instances increase its size to 1.76 Mb, 10^4 instances to 5.54 Mb, $5*10^4$ instances to 22.34 Mb and 10^5 instances to 43.34 Mb. The agent with 10^5 or 100000 instances was running on a Sparc 5 with 64 Mb of RAM, which ensured there was no performance degradation when accessing those objects since the whole process image was kept in core memory. An agent with half a million or even one million MOs is possible through a computer with 256 and 512 Mb of RAM respectively, assuming all the MOs need to be kept in core memory. In general though, "inactive" MOs can be put on secondary storage through the persistency service, reducing the requirements for the required core memory.

Another important aspect worth measuring is the memory overhead of open associations. One of the arguments behind SNMP is its connectionless nature, which allows centralised management of thousands of network nodes. While this is not exactly a requirement in the TMN because of its distributed hierarchical nature, it is worth investigating if connection-oriented management scales. Experiments involved establishing multiple associations between a simple manager and agent. The memory overhead of an existing association was found to be 9.2 Kb for the initiator and 7.2 Kb for the responder. The asymmetry of the observed overhead has to do with the way the ISODE infrastructure handles associations since it was also observed by repeating the experiment at the ACSAP point. Assuming that the manager initiates those associations, the overhead is 920 Kb or roughly 1 Mb for every 100 associations. This does not pose a difficult problem, since a manager managing 1000 devices would be 10 Mb bigger at runtime. A particular issue though is that UNIX systems allow a only a maximum number of file descriptors open per process at a time, typically 64 or 256. Dealing with more descriptors necessitates to re-build the UNIX kernel.

Description	Size (Kb)
Minimal TMN application size	1400
Overhead of a "typical" MO class with no behaviour	10-15
Overhead of the uxObj class	18
Overhead of the uxObj instance as in Code 3-14	0.42
Overhead of an association (initiator)	9.2
Small DSA size	1500

Table 3-7 Summary of Q_3 Memory Overheads

Table 3-7 shows a summary of the application size overheads. In practice, fairly complex TMN applications like the ones implemented in the ICM project [Gri95][Gri96a] had a typical size between 3 and 5 Mb at runtime. As another benchmark, an agent implementing the OSI version of the Internet SNMP MIB-II [Laba91b] in a non-proxy fashion amounts to 3.2 Mb at runtime. The size of the equivalent SNMP agent is 1 Mb i.e. one third of the OSI-SM agent size. While it still does not make sense to manage devices such as video-recorders, set-top boxes, modems and repeaters with full Q_3 agents, the application sizes described above pose hardly a problem for typical telecommunication devices e.g. switches, multiplexors, exchanges, intelligent peripherals, etc. In fact, the size of a TMN OS with useful intelligence can be actually smaller than the size of a word-processing program on a desktop or laptop PC.

We will finally examine the application size of the QUIPU Directory Service Agent (DSA), which is used for dynamic address resolution and location transparency. The typical size of a DSA which serves the needs of the management environment, i.e. *not* a general purpose DSA with tens of thousands of directory objects, amounts to around 1.5 Mb at run time. This is reasonable and is in fact very similar to the minimal size of a TMN OS.

3.8.3 Response Times

In this section we examine the response times of management operations. Despite the fact that the TMN does not operate with the same stringent real-time requirements as the control plane, it still needs to react relatively fast to evolving network conditions. As such, it is important that the supporting infrastructure exhibits good performance characteristics.

SAP	**Sparc 5 - Sparc 20 (msecs - %)**		**486/75 - 486/100 (msecs - %)**	
	Connection Establishment	**Connection Release**	**Connection Establishment**	**Connection Release**
TSAP	10 - *25.6*	0.9 - *13.8*	19.1 - *18.7*	2 - *16.7*
SSAP	15.5 - *39.7*	3 - *46.2*	33.1 - *32.4*	6.25 - *52.1*
PSAP	19 - *48.7*	3.5 - *53.8*	51 - *50.0*	7.4 - *61.5*
ACSAP	22 - *56.4*	5 - 76.9	64.2 - *62.9*	9.5 - 79.2
MSAP"	37.7 - *96.7*	6.25 - *96.1*	100 - *98.0*	11.6 - *96.7*
MSAP	39 - *100*	6.5 - *100*	102 - *100*	12 - *100*

Table 3-8 Response Times for Association Establishment and Release

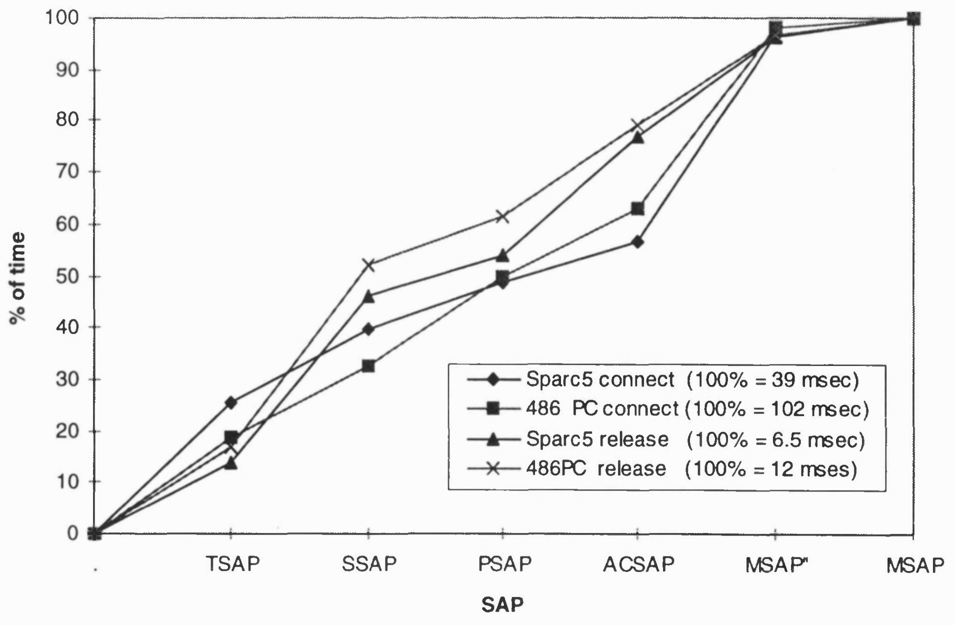


Figure 3-20 Response Times for Association Establishment and Release

We will examine first the performance aspects of establishing and releasing management associations. Table 3-8 shows the response times for association establishment and release at the various access points defined before. Both the absolute and normalised response times for the two pairs of computers and relevant environments used for the measurements are shown. Figure 3-20 depicts the same data in a graph form.

In general, establishing an association is expensive because of the various negotiations that need to take place in the session, presentation and application layer. It should be noted that while a transport connection requires only one request-response exchange, session / presentation connections and application associations require two request-response exchanges. Connection establishment times increase almost linearly until the ACSAP point. Establishing an ACSE association takes around 60% of the time of establishing a full Q_3 association with the information specified by CMIP [X711]. The overhead imposed by CMISE is almost 35% of the overall association time and this can be attributed mostly to the overhead of encoding and decoding the relevant negotiation information. The overhead of the proposed TMN application framework, which is reflected by the difference between the MSAP and MSAP" access points, is minimal i.e. around 3-4% of the overall association establishment time.

Association release is much faster in absolute figures i.e. around 15% of the association establishment time. In general, it is much easier to tear something down than it is to build it in the first place! It should be also noted that transport connections are torn down with a single packet exchange while session / presentation connections and applications associations require three packets: request, response and final confirmation packet sent by the initiator. Apart from the fact that closing down the association is faster, Figure 3-20 shows also that the overhead is more or less equally shared at each layer. This is logical since CMIP does not require to pass any information when closing down an association. Finally, the overhead of the proposed TMN application framework is again very small i.e. again between 3-4% of the overall association release time.

Association establishment and release to the DSA for location transparency purposes exhibits very similar response times to those presented above. This makes sense since the protocol stack is exactly the same until ACSE while the Directory Access Service (DAS) [X511] specifies initial association information similar to the one passed across for CMISE associations.

We will examine now the response times for management operations. While the performance of association control is important, it is reminded here that many operations are typically "multiplexed" onto the same association, with associations "cached" and released after a period of inactivity. Given this mode of operation, response time for management operations are more critical. We will use the echo operation described in section 3.8.1 as a benchmark management operation. Table 3-9 shows the response times in the various access points echoing the "hello" string. Figure 3-21 shows the same information in a graph form.

SAP	Sparc 5 - Sparc 20 (msecs - %)	486/75 - 486/100 (msecs - %)
TSAP	2.2 - *18.3*	4.7 - *17.4*
SSAP	3.2 - *26.7*	6.2 - *23.0*
PSAP	4.2 - *35.0*	9.3 - *34.4*
ROSAP	5.5 - *45.8*	13 - *48.1*
MSAP"	10.7 - *89.2*	25 - *92.5*
MSAP	12 - *100*	27 - *100*

Table 3-9 Response Times for an Echo Operation

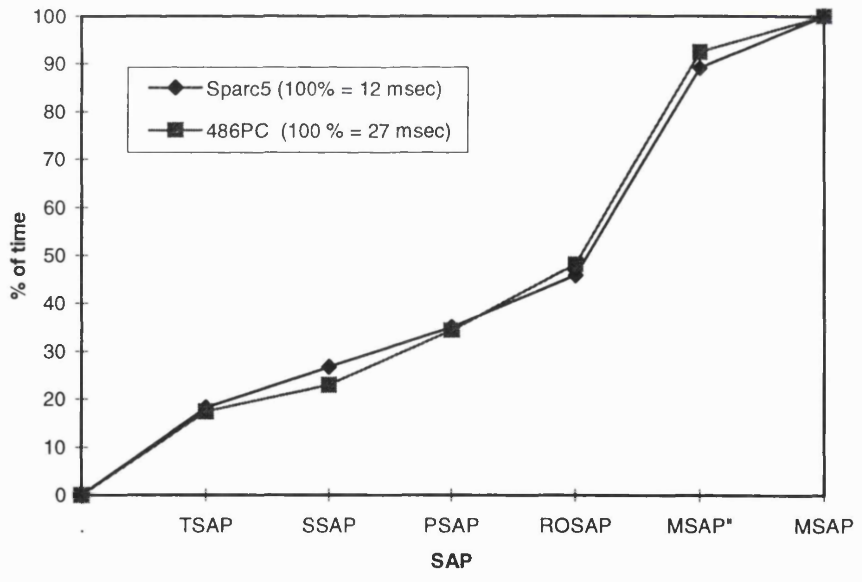


Figure 3-21 Response Times for an Echo Operation

The pattern experienced is similar to that for association establishment. The overhead of the echo operation at the ROSAP point is roughly half of that at the MSAP point. The overhead of the CMISE layer, as reflected by the difference between the MSAP" and ROSAP access points, is around 45% of the overall overhead. This can be explained by the fact that CMIP PDUs are fairly complex ASN.1 structures and a significant overhead is involved in encoding and decoding them.

In general, ASN.1 encoding and decoding in ISODE is dealt with is in two stages, involving substantial memory manipulations in both. While this approach is general and flexible, as discussed previously in this chapter, it seems to incur a substantial performance overhead. [Huit92] claims that ASN.1 performance can be in fact higher than other similar mechanisms such as Sun's eXternal Data Representation (XDR) [Sun87], but their approach is a *one pass* encoding / decoding together and the lightweight encoding rules are used instead of the BER [X209].

The overhead of the proposed TMN application framework is reflected by the difference between the MSAP and MSAP access points, i.e. the overhead of the GMS generic agent and RMIB generic manager infrastructure, and amounts roughly to 10% of the overall overhead. This is a particularly important result, since the same application framework can be mapped over other mechanisms. In Chapter 4, we will discuss such a mapping on OMG CORBA.

While not shown in Table 3-9, using the CMOT instead of the full Q_3 stack results in MSAP response times that are around 15% smaller. In general, both the response time and memory advantages of CMOT are not significant, while Q_3 compliance and interoperability is lost. In summary, it is really not beneficial to use the CMOT stack.

The above experiments at the MSAP" and MSAP access points were conducted with the manager application operating in an asynchronous fashion. This means the request was sent through the asynchronous RMIB API and execution control was passed to the coordination mechanism. The latter passed control back to the RMIB infrastructure when the confirmation packet arrived, which in turn passed the result to the performing managing object.

An interesting finding is that exactly the same request in a synchronous fashion resulted in 7-8% faster response times. While this is initially surprising, it reveals that the cost of the coodination mechanism which was described in section 3.7.2 is not negligible. In addition, there is more context switching. Despite the fact that operations are slightly more expensive when performed in an asynchronous fashion, the performing application may be doing something useful while waiting for the confirmation packet. Of course, this can be also the case with synchronous operations performed in a multi-threaded execution environment.

We will now examine where exactly the overall end-to-end overhead is attributed to. We seeded both the manager and agent applications with instrumentation that took timestamps at various stages of the operation in progress. Having analysed those, the results are very interesting. The procedure of sending the request down the protocol stack in the manager amounts to around 25% (3 msecs) of the overall response time. When the confirmation arrives, it also takes around 25% of the overall response time for the message to ascend the protocol stack.. The same figures for the indication and response messages at the agent amount to 15% of the overall time (1.8 msecs). This means that 25+25+15+15=80% of the overall time (9.6 msecs) is spent for the CMIS messages to descend and ascend the protocol stack. If we add to that the overhead of the asynchronicity and coordination mechanism (1 msec) and the network latency (0.5 msec), we are close to 90% of the overall time (11 msecs) which is the overhead at the MSAP" point. The rest 10% of the overhead (1 msec) is due to the RMIB and GMS infrastructure. The absolute figures mentioned above are for the Sparc 5 - Sparc 20 pair of computers.

What is particularly interesting with these figures is the asymmetry of the overhead of sending and receiving messages in both the agent and manager. Since the difference is not negligible, i.e. 25% vs. 15% of the overall time respectively (or 3 vs. 1.8 msecs), a number of additional measurements were conducted which verified this behaviour. The author cannot find a suitable explanation since the CMIP action request and reply packets are not that dissimilar. Detailed profiling is necessary to see what exactly causes this discrepancy but this was outside the scope of this study.

We will examine now the performance impact of a number of other parameters. We will consider first the size of the data in the management operation. The response times for the echo operation with a string of length 5 ("hello") was shown in Table 3-9. Increasing gradually the size of the echoed string results in an almost linear increase of the response time. The additional overhead is 0.21 msecs or 0.0175% of the overall response time per 100 bytes. The response time increase for amounts of data seems reasonable.

We will consider now the response times using CMIS scoping. Performing an operation on more than one objects through scoping results in a 25% increase of the response time per additional object. This means that 5 scoped operations take the same time as 2 non-scoped ones and this 5:2 ratio holds for other sizes. The 25% overhead per additional object is smaller than the expected overhead according to the above figures for descending and ascending the protocol stack: it should be at least 25%+15%=40% for every additional response. The fact that additional results are sent and received "back-to-back" has positive effects in the overall latency, resulting in smaller times for processing those packets. This behaviour was verified by additional measurements.

We will now investigate the overhead of name resolution, scoping and filtering inside the agent. Name resolution is implemented through a MIT recursive descent and comparison of relative names. As such, it should be a function of the breadth and depth of the MIT, i.e. *f(b,d)*. In order to measure this, we created 5000 uxObj instances and performed various experiments. The additional overhead for searching through 100 MOs until the specified one is found is 1.5%. This means a 15% overhead for searching through 1000 MOs. As discussed in section 3.6, a hash table approach would result in a more or less "flat" delay for locating the object anywhere in the MIT. Even with the current approach though, the overhead is fairly small i.e. an additional 1.5% for searching through 100 objects.

The cost of performing the scope operation within the agent is negligible. This operation simply assembles the right MO pointers by traversing the MIT, starting at the base MO. This is not surprising since assembling pointers through a recursive tree descent algorithm is something that modern processors do very quickly. The same is true for evaluating a filter on a managed object. We tried a number of filters on the uxObj MO and the evaluation time was negligible compared to the end-to-end response time. These filters were evaluated without "refreshing" the attribute values first, so that only the overhead of filtering was measured.

Finally, the cost of accessing the directory for location transparency reasons is very similar to the cost of performing a read or action operation on a managed object. The cost of establishing an association to the DSA, retrieving the presentation address of a TMN application, establishing an association to the latter and performing a management operation takes in total 40 + 10 + 40 + 10 = 100 msecs using the Sparc 5 / Sparc 20 setup. It should be noted though that this sequence of operations takes place only the first time. The presentation address of the target application is "cached" in the performing application while the same is true for the association to that application. The second operation will take in this case around 10 msecs.

3.8.4 Packet Sizes

We finally consider the sizes of management packets. These include the payload measured over the TCP, as explained in section 3.8.1.

Table 3-10 and Table 3-11 show the sizes of the packets exchanged for establishing and releasing connections at the various service access points. As already mentioned, from the session layer and above connection establishment requires 4 packet exchanges (2+2) while connection release requires 3 packet exchanges (2+1). It should be also mentioned that none of those packets are "piggy-backed" on the TCP packets for connection establishment and release, so we should count another two packet exchanges with no data. In summary, upper layer connection establishment requires 6 while connection release requires 5 overall packet exchanges.

SAP	Data sent (bytes / packet)	Data Received (bytes / packet)	Total (bytes)
TSAP	21	17	38
SSAP	18 29	14 31	92
PSAP	18 40	14 42	114
ACSAP	18 132	14 130	294
MSAP	18 140	14 138	310

Table 3-10 Packet Sizes for Connection Establishment

SAP	Data sent (bytes / packet)	Data Received (bytes / packet)	Total (bytes)
TSAP	11		11
SSAP	12 11	9	32
PSAP	12 11	9	32
ACSAP	28 11	25	64
MSAP	28 11	25	64

Table 3-11 Packet Sizes for Connection Release

A graph depiction of the total data in Table 3-10 and Table 3-11 is shown in Figure 3-22. ACSE adds another 180 bytes on top of the presentation layer connection setup, which requires the exchange of 114 bytes. CMISE adds another 16 bytes of initial information over ACSE. A substantial amount of traffic is required to setup a management association: 6 packets and 310 bytes in total. Connection release requires the exchange of 5 packets but the amount of overall data exchanged is much smaller, 64 bytes in total. In summary, management association establishment and release require a significant number of packet exchanges. In addition, association establishment incurs also a significant traffic overhead.

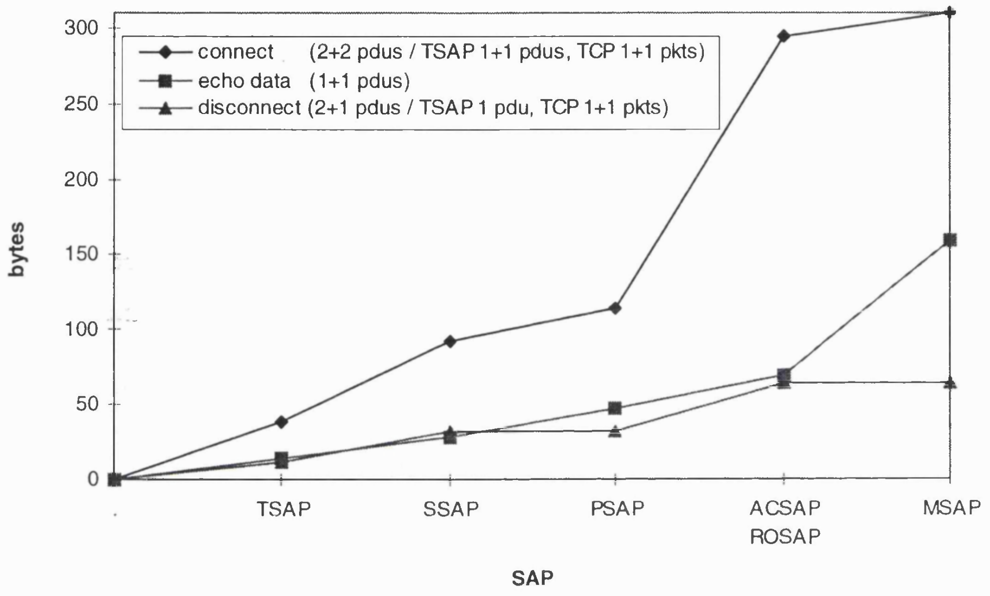


Figure 3-22 Connection Establishment, Release and Echo Operation Packet Sizes

SAP	Data sent (bytes)	Data Received (bytes)	Total (bytes)
TSAP	7	7	14
SSAP	14	14	28
PSAP	24	24	48
ROSAP	34	36	70
MSAP	72	88	160

Table 3-12 Packet Sizes for the Echo Operation (zero length string)

Table 3-12 shows the packet sizes at the various SAPs for the echo operation, echoing an empty string. Figure 3-22 shows also the overall data exchanged in graph form, together with the same data for connection establishment and release. The echo operation requires a request and a response packet in all the SAPs. In this case, the CMISE layer (difference between the MSAP and ROSAP access points) adds a substantial amount of data because of the parameters in the relevant PDUs. It should be noted that both the action request and response PDUs include the name of the addressed object which is "uxObjId=null". The overhead of that particular distinguished name component is exactly 16 bytes, including 4 bytes for the "null" string. In general, the request and response packet sizes will be larger for objects located "deeper" in the MIT.

Retrieving the wiseSaying attribute with value "hello world" of the uxObj instance incurs a request packet of 67 bytes and a response packet of 100 bytes. Retrieving all the attributes of the uxObj instance as depicted in the Code 3-14 caption but with the relative name "uxObjId=null" results in a request packet of 62 bytes and a response packet of 142 bytes.

3.8.5 Summary

In this section we looked at the performance aspects of the proposed object-oriented development environment in terms of application size, response times and packet sizes.

In subsection 3.8.2 we looked at TMN application size. A summary of the relevant overheads were presented in the Table 3-6. The minimal TMN application is about 1.4 Mb at run-time while example applications with fairly complex behaviour and a number of object were 3-5 Mb at run-time. The impact of additional object instances depends on the number and size of relevant attributes and other private instance variables. A typical overhead of an instance is smaller that 1 Kb, which means that applications with tens of thousands of managed objects are feasible. While the minimal application size, which effectively represents the overhead of the infrastructure, is not small, it is hardly a challenge for today's computers and typical memory sizes. In fact, such applications run happily on the author's laptop computer with 8 Mb of RAM and an Intel 486 processor.

In subsection 3.8.3 we looked at response times and presented results for two types of fairly modest computers. Association establishment is around four times more expensive than performing a simple management operation and this highlights the fact that associations should be typically "cached" and released only after a period of inactivity. Absolute times for association establishment are around 40 msecs for the relatively powerful computers and around 100 msecs for fairly old technology PCs. Absolute times for simple management operations are around 10 and 40 msecs respectively. While these absolute numbers are only meaningful when compared with similar operations in other frameworks, they confirm the subjective feeling of fast operation for the overall framework, which is also acknowledged by other researchers. For example, [Deri97] compares the performance of a number of management environments, including

OSIMIS, and characterises it as "performant" while it characterises typical commercial TMN platforms as "partially performant, requiring powerful hosts and a lot of memory".

What is more important is that the detailed measurements at various service access points and within management applications reveal that the main performance overhead is due to the protocol stack rather than the proposed TMN application framework. This is particularly important since the protocol stack can be replaced with other, more lightweight stacks. Alternative mappings for CMIP were discussed in section 3.3.2.4 while in the next chapter we will present a totally different mapping over OMG CORBA and will evaluate its performance.

Finally, in subsection 3.8.4 we looked at packet sizes. Association establishment seems to incur a significant amount of traffic, involving 6 overall packet exchanges. On the other hand, the packet sizes of management operations seem relatively reasonable, considering in particular the fact that OSI protocols and the BER are used. In subjective terms, the author was expecting much higher figures before conducting the experiments. It should be finally mentioned that packet sizes are pertinent to the *specification* of the OSI-SM/TMN framework while application sizes and response times are pertinent both to the specification and the proposed software *realisation* framework, including the ISODE OSI upper layer environment.

It should be finally mentioned that the performance evaluation in this section was not exhaustive. A detailed performance analysis of an OSI-SM platform and possibly its comparison to emerging distributed object frameworks such as OMG CORBA could be the subject of a whole thesis. The relevant study had as target to show the salient performance characteristics of the proposed OSI-SM / TMN software framework. Despite its limited scope, this work is significant since it shows that the proposed framework is performant, as detailed above. In addition, it is the very first of this type i.e. no relevant work can be found in the literature.

3.9 Validation

In the previous sections of this chapter we presented various aspects related to the object-oriented realisation of the TMN framework. Throughout those sections we explained how the proposed solutions contribute towards the goals of this thesis, validating the relevant assertions made in the beginning. In this section, we summarise the already presented validation aspects and validate further the proposed framework against additional criteria.

3.9.1 The Proposed Environment as an Object-Oriented Distributed Framework

We will examine first if the OSI-SM / TMN architectural framework together with the proposed realisation framework satisfy the key properties of object-oriented distribution frameworks, as identified in section 3.2.2. We re-iterate through those properties below and examine if and how these are satisfied.

- *An abstract object-oriented specification language that supports inheritance and polymorphism.* GDMO satisfies those requirements with the only drawback that actions and notifications do not map in a satisfactory fashion to abstract remote method calls. The modifications proposed in section 3.5.4.3 rectify this drawback. Of course it is not claimed here that GDMO is a general purpose O-O specification language in the same fashion as CORBA IDL, we comment further on this at the end of this section.

- *Mappings of that abstract language to multiple object-oriented and procedural/modular programming languages.* We have shown mappings of GDMO to C++ in this section that are general enough to be mapped onto other O-O programming languages e.g. Smalltalk, Java. We have also shown a manager mapping to Tcl/Tk which is a procedural scripting language.

- *User-friendly APIs that hide communication and protocol details.* Such APIs were presented through the ASN.1, RMIB/SMIB and GMS O-O concepts and infrastructures. It should be noted that powerful CMIS access aspects (scoping, filtering) and fine-grain event reporting are still available in a natural, "harness and hide" fashion.

- *Dynamic access facilities that obviate the need for static (i.e. pre-compiled) knowledge of object specifications in client applications.* Both the RMIB and weakly-typed SMIB APIs support this requirement. ASN.1 manipulation is static in OSIMIS, i.e. generic applications need to be re-linked with new types, but we have explained how this can be avoided through a data-driven approach in section 3.5.6.

- *Good performance and scalability so that distribution is encouraged and exploited.* The performance was shown to be good from a number of perspectives in the previous section. In addition, both the average application and object sizes are reasonable. Though scalability was not explicitly addressed from a global system perspective, the performance and size figures do not reveal any major deficiencies.

- *Openness in terms of both standard APIs and communication protocols.* The O-O APIs presented in this section could support "horizontal" openness and portability if they were standardised. The NMF TMN/C++ [Chat97] family of APIs, which has re-used many of the concepts proposed here, will be the relevant industrial standard. "Vertical", on-the-wire openness is provided through the Q_3 protocol stack [Q3].

- *Distribution transparencies, in particular access and location.* Access transparency is supported through the Q3 protocol stack, including ASN.1 [ASN1] and BER [BER] for data representation. Finally, location transparency is provided through the OSI directory [X750] mechanism discussed in Chapter 2 and the realisation model proposed in the previous sections of this chapter.

In conclusion, these properties are largely satisfied. The one that is only partly satisfied concerns the *standard* mapping to multiple programming languages. The NMF TMN/C++ API covers only the mapping to C++. Standard mappings to other object-oriented and procedural programming languages are not currently addressed. In comparison, OMG CORBA [CORBA] offers mappings to C, C++, Smalltalk and Java. The approach in this chapter though has shown that "horizontal", O-O distributed system-like APIs are feasible for harnessing the OSI-SM/TMN power and complexity. Standard APIs may be specified for a number of programming languages in the future.

It should be also noted that the OSI-SM/TMN model is *not* a general distributed systems model but targets distributed management systems. It is a composite model in which managed objects are clustered together in "ensembles", handled by applications in agent roles as it was explained in Chapter 2. Despite the fact we have used distributed systems examples to demonstrate the distributed system-like nature of the proposed realisation model, e.g. those involving the simpleStats class, these are degenerate cases in which the ensemble becomes essentially a single object. On the other hand, the fact that such cases can be accommodated reinforces the validity of the assertion that the OSI-SM / TMN framework can be realised in a distributed system like fashion.

3.9.2 Object-Oriented Support for TMN Operations Systems

In Figure 2-22 of Chapter 2 we presented a functional decomposition of the OSF which is the fundamental TMN building block. Having presented the object-oriented realisation framework in the previous sections, we present in Figure 3-23 a concrete engineering object-oriented decomposition of a TMN Operations System, as supported by the proposed OSI-SM / TMN realisation framework. This is a *generic* decomposition which shows how the various facilities of the proposed infrastructure are used.

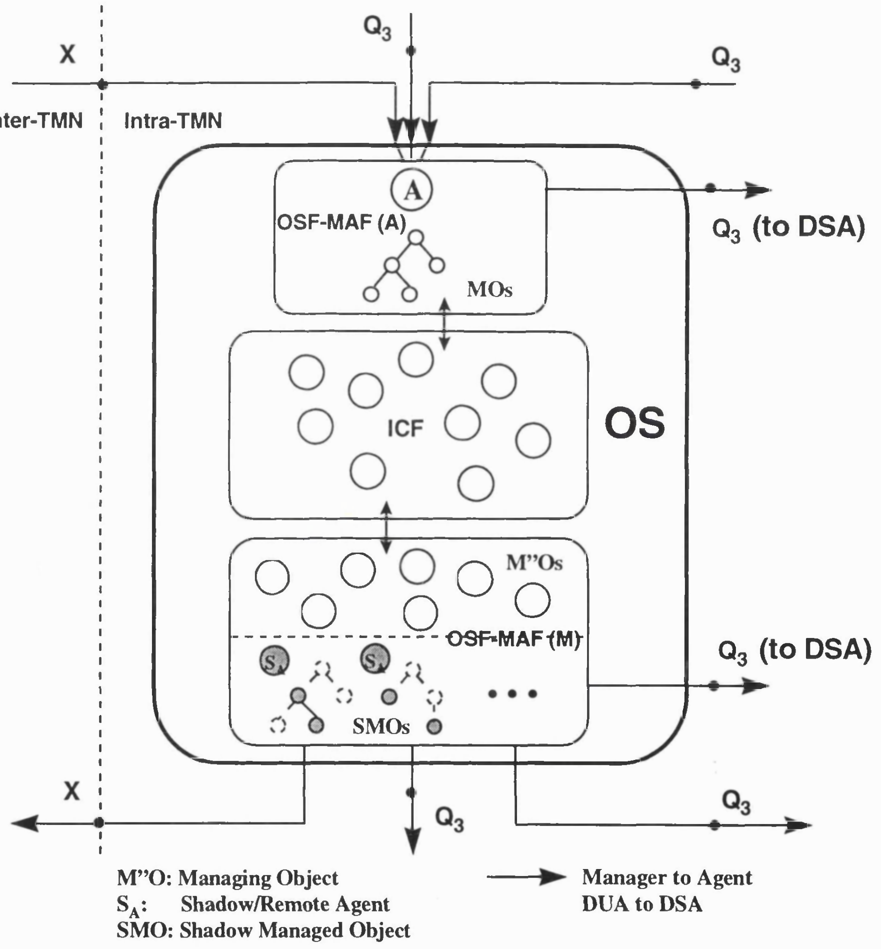


Figure 3-23 Object-Oriented Decomposition of a TMN OS

The Management Application Function in agent role (MAF-A) is realised through the generic agent infrastructure, the Generic Management System (GMS). The managed object classes available across the Q_3 and X interface are compiled through the GDMO compiler and enhanced with behaviour, as was described in section 3.6. Access to them is provided through the agent object instance, which encapsulates the interoperable protocol stack and provides the various CMIS services, including scoping, filtering, synchronisation and event reporting. The authentication, stream integrity and confidentiality services are provided by the protocol stack, either by the presentation layer [GULS] or by CMISE/ROSE [Bhat96]. The access control service is provided by an Access Decision Function (ADF) object instance which is also encapsulated by the agent object [Pav96b]. The MAF-A contains pre-compiled knowledge of the managed object classes of the Q_3 and X ensembles, including also event reporting, logging and any other SMFs required by the relevant specifications e.g. metric monitoring [X739], summarisation [X738], performance management [Q822], fault management [Q821], etc. Any differentiation between the X and Q_3 interfaces offered to peer and superior OSs is provided through access control [X741].

The Management Application Function in manager role (MAF-A) is realised through the generic manager infrastructure, the Remote MIB (RMIB) and/or Shadow MIB (SMIB). The managed object classes of the Q_3 and X information model in superior and peer systems need to be compiled in order to produce the information necessary for the relevant repository. In addition, if a static, strongly-typed SMIB model is followed, specific SMO classes are produced and possibly enhanced with behaviour as it was described in section 3.5.4. Typically one Shadow/Remote agent and associated SMOs exist for every accessed TMN, as shown in Figure 3-23. Various managing objects implement the OS managing intelligence. A part of the application's management intelligence is realised by various objects implementing the Information Conversion Function (ICF). These interact both with managed and managing objects while their functionality is typically triggered by operations to managed objects "downwards" or through received event reports and results of access to subordinate or peer systems "upwards".

Directory access is required in order to update the directory about the location and capabilities of the OS and in order to discover the location and capabilities of other TMN. It is reminded here that directory access facilities are integrated with the CMISE API, as described in section 3.3.2.3. These are used by the MIB agent, which updates the directory about the location, capabilities and supported managed objects classes and instances for this OS. They are also used by the various RMIB and subsequently SMIB agents (a SMIB agent always contains a RMIB agent) which discover the location and capabilities of other applications.

Finally, such an application can be organised either in a multi-threaded fashion or in a single-threaded fashion as described in section 3.7. In the latter case, manager access to peer or subordinate systems should be asynchronous so that the application can still be active while waiting for a result or results.

3.9.3 Support for Peer-to-Peer Interactions

Another interesting point related to the validation of the proposed framework is natural support for peer-to-peer interactions. As already mentioned in section 2.4.5 of Chapter 2, the inherent asymmetry of the manager-agent model can be a limitation if the relevant engineering concepts separate completely the manager and agent aspects. This is certainly not the case with the proposed realisation model. Though Figure 3-23 presents a hierarchical internal decomposition with the MAF-A, ICF and MAF-M separation, this need not be so if particular peer-to-peer exchanges require it. For example, a managed object and managing object can be realised by the same object instance, acting in both roles. In this case, an operation to that instance may trigger an operation in the opposite direction, with the managed object becoming instantly a managing object. In fact, such operations may also pass the global distinguished name of the invoking object as an action parameter so that the accessed object can perform an operation in the opposite direction. This bears an exact analogy to the passing of object references in distributed system frameworks such as OMG CORBA, as discussed in Chapter 4.

3.9.4 Implementability in Terms of the Overall Required Software

Since one of the objectives of this thesis is to show the feasibility and implementability of the OSI-SM/TMN framework, it is interesting to investigate how much software in terms of lines of code is necessary to provide this type of flexible object-oriented environment. We will assume that there exists an upper layer OSI stack, including ACSE, ROSE, DASE implementations and a procedural ASN.1 compiler, which is what most OSI infrastructures provide.

Table 3-13 shows the size of the generic parts of the infrastructure in lines of code. This is in C++ unless it is indicated otherwise. The grand total is 55000 lines which seems reasonable for the relevant functionality and demonstrates the feasibility of the approach. Of course, turning this prototype this into production software would require a substantial amount of additional development. This has been done though by more than one commercial vendors of TMN infrastructure, validating further the relevant concepts.

It should be noted that the Table 3-13 includes only the absolutely necessary TMN platform facilities. As such, it does not include the generic MIB browser [Pav92a] (5000 lines), the generic CMIS/P to SNMP adapter [McCar95] (12000 lines together with the SNMP SMI to GDMO compiler) and the lightweight, secret key based authentication, integrity and confidentiality services [Bhat96]. It should be finally mentioned that the author developed around 65% of the amount of software mentioned in the Table 3-13.

Component	**Lines of code[5] and description**
CMISE	10000 (in C, includes also higher-level support functions)
X.500 access for X.750	3000 (in C)
O-O ASN.1 API / compiler and X.721 attributes	10000 (1000 the compiler in gawk, 2000 the generic ASN.1 classes, 8000 the rest)
GDMO compiler	10000 (8000 the compiler, 2000 the code generating scripts)
GMS Agent infrastructure	10000 (6000 the "kernel", 1500 event reporting, 1500 logging, 1000 access control)
RMIB Manager infrastructure	3000
Tcl-RMIB Manager infrastructure	3000
Generic command-line manager programs	3500 (these are required as debugging tools)
Coordination support	1500
Generic library classes	1000 (string, list, array)
Total	55000

Table 3-13 Amount of Software in the OSIMIS TMN Platform

3.9.5 Further Validation

Finally, the true validation and verification of the proposed framework has been achieved through its use to develop and deploy experimental TMN systems and through its use for additional research by the author and many other researchers. Appendix A provides a short description of the known research work that has been based on the proposed object-oriented TMN platform.

[5] In C++, unless mentioned otherwise.

3.10 Summary

3.10.1 Overview of this Chapter

In this chapter we presented first an introduction to object-oriented design and development principles, followed by the identification of desired properties of object-oriented distributed frameworks. These were used later to measure against them the proposed object-oriented TMN platform. The same properties will be also used in Chapter 4 to measure against them ODP-influenced distributed object technologies.

We subsequently looked at issues behind the realisation of the Q_3 protocol stack. We discussed aspects of realising ROSE and CMISE over the presentation service and presented possible design policies for a CMIS API. The latter is completely hidden behind higher level object-oriented infrastructures but is the fundamental building block for those. We also presented two lightweight mappings for CMISE that operate directly over the transport service and use string encodings for attribute, action and notification values.

We then discussed aspects behind object-oriented ASN.1 manipulation and presented a design that uses polymorphic principles. ASN.1 types map to C++ classes which are automatically produced by an O-O ASN.1 compiler and may be further customised by the implementor. Additional classes provide support for the ASN.1 ANY and ANY DEFINED BY constructs. These generic classes are used extensively in the higher-level managed and agent infrastructures and APIs.

We then discussed extensively aspects behind the realisation of object-oriented manager infrastructures and proposed two models, the Remote MIB and Shadow MIB. These hide completely underlying protocol details but provide access to powerful CMIS aspects such as multiple object access through scoping and filtering and fine-grain event reporting. A mapping of the RMIB infrastructure to the scripting Tcl language was also presented. This may be used for applications with GUIs, i.e. TMN WS-OSs, because of the associated Tk graphical toolkit. We also discussed aspects of the management information repository which represents the necessary "shared knowledge" between agent and manager applications that support a Q_3 interface.

A discussion on the aspects behind the realisation of object-oriented agent infrastructures followed. We presented a generic agent architecture known as the Generic Managed System which separates service and protocol processing from the managed objects. We also proposed a mapping of the GDMO abstract language to C++ in a fashion that behaviour is added through the redefinition of polymorphic methods, shielding the implementor from unnecessary details. Stub managed objects are produced through a GDMO/ASN.1 compiler and are augmented with behaviour. We also discussed how the "difficult" aspects of OSI-SM can be implemented using O-O principles that harness and hide the relevant complexity.

We then discussed briefly issues behind synchronous vs. asynchronous remote execution models. Synchronous models are more user-friendly but require support for multithreaded execution and concurrency control. Asynchronous models are less user-friendly but do not necessarily require support for multithreading. In either of the two models, a coordination mechanism is needed to "dispatch" activities within a complex application implemented as a single operating system process. An object-oriented coordination infrastructure was presented which shields application objects from the underlying complexity.

We then presented a performance analysis and evaluation of the proposed object-oriented environment, addressing program size, response times and packet sizes. It was shown that the overheads of TMN applications are reasonable and much smaller than widely believed, though this is a relatively subjective judgement. In Chapter 4 we will perform similar measurements for CORBA. These will put the measurements for the OSI-SM based framework into perspective.

Finally, a validation of the proposed framework was presented by examining it against the properties of object-oriented distributed frameworks identified in the beginning. We also explained how the proposed O-O infrastructure can support the various aspects of a TMN OS and considered the amount of software required to build such an infrastructure. Since the ultimate validation of the proposed environment was achieved through its use for further research, design and development, relevant efforts are presented in Appendix A.

3.10.2 Research Contribution

The key research contributions in this chapter are the following:

- The identification and discussion of the general issues in realising transaction-based OSI upper layer protocols that rely on ROSE and ASN.1 in section 3.3.2.
- The detailed presentation of various issues and alternative approaches in realising CMISE APIs that include association control and location transparency features in section 3.3.2.3.
  - ⇒ The validation of the procedural, asynchronous, "lowest common denominator" CMISE API approach through the author's design and implementation of the OSIMIS CMISE. This has been used world-wide and influenced both commercial products and subsequent API standards.
- The presentation of issues behind alternative lightweight mappings for the CMIP protocol and the specification of two approaches of a "string-based" CMIS/P that can operate directly over various reliable transport mechanisms in section 3.3.2.4.
- The presentation of the issues behind object-oriented, polymorphic ASN.1 manipulation in OSI upper layer infrastructures and the specification of the generic classes realising the relevant API in section 3.4.
  - ⇒ The validation of the O-O ASN.1 approach and API through the author's design and implementation of the relevant generic classes and of an O-O ASN.1 compiler that generates specific classes for ASN.1 types. The O-O ASN.1 is used in the other OSIMIS high-level O-O APIs.
- The presentation of the issues and the design behind object-oriented, polymorphic manager infrastructures and the identification of two major approaches in section 3.5:
  - * An agent-based approach of dynamic, weakly-typed nature which was termed the Remote MIB (RMIB).
  - * A managed object-based approach which can be both dynamic / weakly-typed or static / strongly-typed, termed the Shadow MIB (SMIB). This includes the mapping of GDMO to O-O programming languages from a client or manager viewpoint.

  * The specification of the relevant classes and APIs and the demonstration through examples of their flexibility, user-friendliness and economy in terms of lines of code required for distributed operations, resulting in a rapid system development cycle.
  * The specification of a string-based RMIB API and its integration in the Tcl/Tk scripting language.

  ⇒ The validation of both approaches through the design and implementation of the relevant infrastructures. The RMIB and Tcl-RMIB have been part of OSIMIS and have been widely used for prototype TMN system development. Both the RMIB and SMIB infrastructures have influenced products and subsequent API standards.

- The identification of a logical "bug" in the CMIS/P m-event-report primitive which makes difficult to demultiplex event reports and pass them to the right managing object within a manager application. This was revealed through the validation of the manager aspects of the OSI-SM framework through the RMIB / SMIB infrastructures. The subsequent proposal of a CMIS/P modification that overcomes this limitation was presented in section 3.5.3.2.

- The identification of an inefficiency in the GDMO Action template which results in non-natural mappings to object-oriented programming languages. This was revealed through the validation of the framework through the RMIB / SMIB infrastructures. The subsequent proposal of a GDMO modification that overcomes this limitation in section 3.5.4.3.

- The presentation of the issues and the design behind object-oriented polymorphic agent infrastructures in section 3.6:

  * The presentation of an object-oriented architecture which separates service and protocol aspects from the managed objects - the Generic Managed System (GMS).
  * The presentation of a GDMO to C++ mapping which uses only single inheritance and can be also provided in languages such as Smalltalk and Java.
  * The presentation of a polymorphic managed object class API which can be used to add behaviour by redefining the relevant polymorphic methods.
  * The presentation of three different approaches for maintaining consistency between managed objects and associated resources.

  - * The discussion how the "difficult" OSI-SM aspects can be provided, including name resolution, scoping, filtering, atomicity, allomorphism and persistence.
  - * The discussion on how event reporting, logging and the rest of the OSI SMFs can be supported in agent environments.
  - ⇒ The validation of all those concepts through the design and implementation of the GMS infrastructure. This has been widely used for prototype TMN system development and influenced commercial products and subsequent API standards.

- The demonstration that the OSIMIS O-O manager and managed object APIs bear a lot of similarities to those of distributed object frameworks such as OMG CORBA which were developed later. This shows that the complexity and power of the OSI-SM/TMN model can be harnessed behind O-O platform APIs that support abstractions similar to those of emerging distributed object frameworks.

- The demonstration that the perceived limitations of the OSI-SM manager-agent separation can be overcome through O-O realisation infrastructures that allow objects to take managed or managing roles at any time. This supports the natural realisation of peer-to-peer interactions for TMN OSs.

- The demonstration that the proposed object-oriented TMN realisation framework has modest requirements in terms of memory resources and it has good performance characteristics, even with modest computing infrastructures. In addition, the generated packet traffic from the supporting OSI protocols is reasonable. These conclusions were drawn through a detailed performance analysis and evaluation in section 3.8 which is the first of this type i.e. there is no similar work in the literature.

- The demonstration through the whole approach that the TMN is not a pile of complex and difficult to implement recommendations. On the contrary, the relevant O-O methodology and specification aspects lend themselves naturally to object-oriented realisation through the "harness and hide" principles presented in this chapter. The result is a powerful, flexible, performant and easy-to-use distributed environment.

- Finally, a more general contribution to telecommunication problem solving which moves away from the protocol-based "bits-and-bytes" approach of the past towards general-purpose, easy-to-use distributed computing environments that satisfy the relevant requirements of both interoperability and software openness.

4. Mapping the OSI-SM / TMN Model Onto Emerging Distributed Object Frameworks

4.1 Introduction

Chapter 4 of this thesis examines the possibility of using emerging distributed object technologies as the basis for the TMN. Since the early inception of Open Distributed Processing, a number of related technologies tried to provide a uniform and ubiquitous framework for building distributed applications. The latest and most powerful of those technologies is OMG CORBA, which can be thought as the pragmatic counterpart of ODP. Since the TMN is a large scale distributed system, it is valid to consider its mapping onto CORBA, considering the latter as the representative distributed object technology. This implies replacing OSI-SM and the OSI Directory with OMG CORBA as the base technology for the TMN. The relationship between OSI-SM and OMG CORBA has attracted considerable attention from the research community in recent years. In this chapter we propose a solution that maintains the expressive power of OSI-SM and provides a smooth migration path towards a CORBA-based TMN.

A key motivation for using CORBA in TMN environments is the following. OSI-SM was conceived as an object-oriented management technology in the absence of a general purpose distributed object-oriented framework. OMG CORBA provides exactly such a framework and it is likely that it will be used in the future for supporting advanced telecommunications services e.g. those conforming to the emerging Telecommunications Networking Information Architecture (TINA) [TINA]. Using OSI-SM to manage distributed components of those services will result in a discrepancy of technologies: one technology for the service operation and control, i.e. OMG CORBA, and another technology used for managing the relevant service components, i.e. OSI-SM. OMG CORBA may provide the unifying framework, resulting in economies of scale.

Additional motivations for using OMG CORBA in TMN systems are the following. CORBA provides a superior distribution paradigm, in which every object could be potentially distributed. In OSI-SM only whole agent applications can be distributed. CORBA performance could be potentially better than OSI-SM due to a more lightweight protocol stack. Finally, CORBA exhibits multiple O-O programming language mappings while both OSIMIS [Pav93b] and the NMF TMN/C++ [Chat97] support mainly C++ APIs. On the other hand, OSI-SM exhibits a more expressive object model and superior object access and event dissemination mechanisms, so the mapping between the two presents a difficult technical challenge.

This chapter is organised as a "super-chapter", in a similar fashion to chapters 2 and 3 of this thesis. Related research work is presented throughput the various sections, in a similar fashion to previous chapters.

Section 4.2 describes Open Distributed Processing, which underpins distributed object technologies. This introduction serves as state of art but also goes further, positioning OSI-SM and TMN in the ODP context and examining them from the ODP viewpoints.

Section 4.3 describes different incarnations of ODP-based technologies, namely ANSA, the OSF DCE and OMG CORBA and considers their suitability as TMN technologies. This is again more than a state of the art description, presenting those technologies as candidates for the TMN and examining them against the properties of distributed frameworks identified in Chapter 2.

Section 4.4 considers in detail how CORBA can be used as the base technology for the TMN. A pure ODP-oriented approach is presented first, which is practically unfeasible with the current state of CORBA technology. Then a more pragmatic mapping approach follows, starting from a minimal solution and adding gradually features so that the full power and expressiveness of OSI-SM is recaptured. This mapping represents the key research contribution of this chapter and has been partly validated through implementation.

Section 4.5 presents a performance analysis and evaluation of the CORBA-based framework, trying to identify the performance characteristics of CORBA and potential advantages over OSI-SM. This section though does not go though into the same level of detail as the relevant OSI-SM analysis in Chapter 3.

Finally, section 4.6 presents a summary and highlights the research contributions of this chapter.

4.2 Open Distributed Processing

The ISO/ITU-T Open Systems Interconnection (OSI) and the IETF standardisation efforts addressed the problem of heterogeneous system interconnection through standardised protocols. The application layer in both frameworks addressed partly the needs of distributed application development through the OSI ROSE [X219] and ASN.1 [X209] and the Internet Sun RPC [Sun88] and XDR [Sun87]. Both ROSE and RPC systems are object-based as opposed to object-oriented. In addition, ROSE is not a software framework and as such it does not support application software portability through standard APIs.

Towards the mid to late eighties it became clear that the development and deployment of distributed applications could be facilitated by software infrastructures with standard APIs, in addition to the required standard protocols for interoperability. The Advanced Networked Systems Architecture (ANSA) [ANSA89a][ANSA89b] was a research and development effort towards a distributed software platform which aimed to provide well-defined APIs for application components, facilitating their rapid development and achieving application portability. In fact, ANSA was more than a distributed software environment: it introduced a set of concepts for distributed systems and also influenced the ISO/ITU-T activity on the standardisation of Open Distributed Processing (ODP) [ODP].

It should be noted that the ISO/ITU-T OSI Systems Management framework was the only OSI application layer effort that adopted a fully object-oriented information specification approach. The realisation framework presented in Chapter 3 proposes a software infrastructure that addresses the issues of rapid application development and software portability through well-defined APIs. This environment has a lot in common with distributed software frameworks based on ODP principles, as it was explained in Chapter 3. Both ANSA and the embryonic ODP framework of the late eighties were important influences for the author. On the other hand, the proposed software framework is specific to object-oriented OSI-SM/TMN systems while ANSA / ODP are more general frameworks, targeting general purpose distributed applications.

4.2.1 The ODP Model

ODP aims to provide both an architectural framework for distributed systems and also associated implementation technology for distributed objects. A standardised software infrastructure will facilitate the integrated support of distribution and will achieve portability and interoperability. The essence of the ODP framework is demonstrated in Figure 4-1. The ODP software platform operates over the native computing and communications environment in the various network nodes, hiding heterogeneity and providing an homogeneous view of the underlying resources. Objects "sitting" on this distributed platform interact with each other while they are shielded from the problems and complexity of distribution through various transparencies.

ODP targets a programmatic interface between objects in *client* and *server* roles and the underlying support environment i.e. the ODP platform. The latter is shown as crossing the boundaries of network nodes in Figure 4-1 because of the fact it hides heterogeneity and provides a uniform view of the underlying resources. Parts of the global ODP platform may be provided by different vendors while objects will get a uniform view through the standardised APIs. Another commonly used term for such a software platform is *middleware*, since it is an enabling technology.

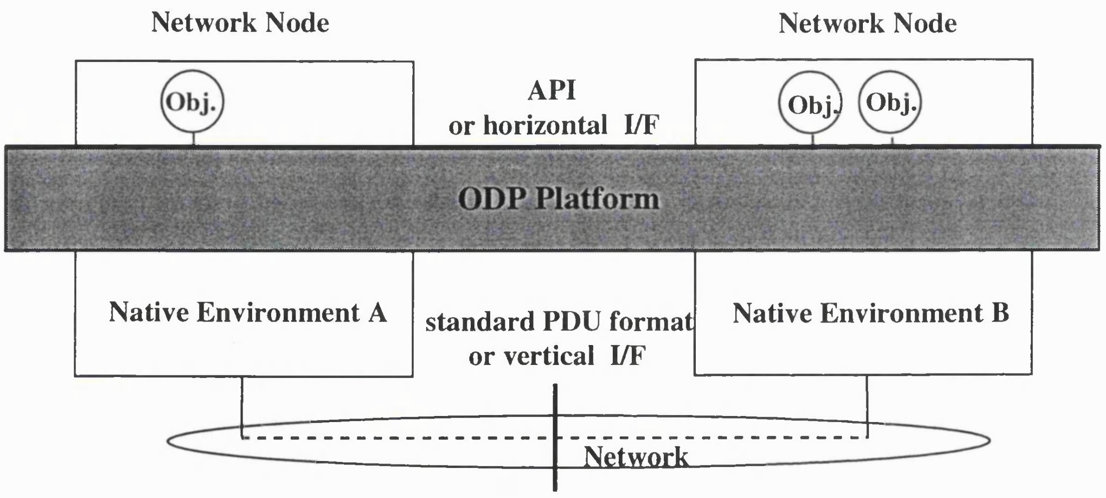


Figure 4-1 The ODP Framework

The approach of standardising APIs for software portability and uniformity is sometimes referred to as "horizontal" standardisation. OSI and the Internet address interoperability through standard protocols which result in agreed message formats; this approach is sometimes referred to as "vertical" or "on the wire" standardisation. ODP systems need to agree on standard protocols in order to interoperate. It should be possible though to port an ODP platform over different underlying communications protocols without any changes to the APIs.

The ISO/ITU-T ODP Reference Model (ODP-RM) [ODP] is split into four parts, all of which use object-oriented concepts. Part 1 [X901] provides the overview, introduces the concept of information distribution, discusses the issues it raises and proposes a number of distribution transparencies to cope with those. Part 2 [X902] is descriptive and introduces modelling, specification and architectural concepts. Part 3 [X903] is prescriptive and provides the concepts, rules and functions a system must deal with in order to be ODP conformant, which are expressed in terms of five *viewpoints*. Finally, Part 4 [X904] attempts to map the modelling concepts of part 2 onto formal description techniques applicable to the five viewpoints.

4.2.2 The ODP Distribution Transparencies

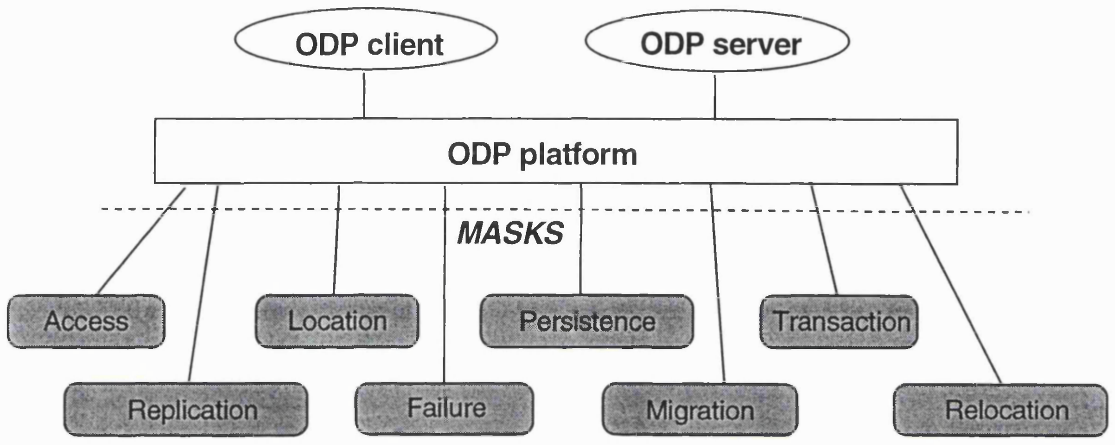


Figure 4-2 ODP Distribution Transparencies

An ODP platform supports the interaction between objects in client and server roles, through *interfaces* supported by the server object i.e. this is an asymmetric relationship in the same fashion to the manager-agent relationship explained in Chapter 2. The ODP platform provides the following transparencies to client objects, shielding them from the problems of distribution:

- *access* transparency, which masks heterogeneity in computer architectures, programming languages, data representations and invocation mechanisms;
- *location* transparency, which masks the topological details when "binding" to distributed server objects;
- *persistence* transparency, which masks the activation and deactivation of a server object with respect to its persistent secondary storage representation;
- *transaction* transparency, which masks the coordination of activities across a number of server objects in order to achieve overall consistency of operations;
- *replication* transparency, which masks the existence of a number of server objects with the same properties used for performance, availability, reliability and fault tolerance;

- *failure* transparency, which masks the failure of objects and their subsequent reactivation or substitution through replication, migration etc.;
- *migration* transparency, which masks the fact that a server object may change location because of performance, security or other reasons; and
- *relocation* transparency, which is related to migration and masks the relocation of a server object while clients are still bound to it and operations may be in progress.

Access, location, persistence and transaction transparencies are not only pertinent to ODP systems. The OSI-SM/TMN object-oriented distributed platform presented in Chapter 3 exhibits access, location and persistence transparencies while transaction transparency could be added through the ISO/ITU-T Distributed Transaction Processing (DTP) [DTP] framework. Failure, replication, migration and relocation transparencies are much more difficult to provide. In fact, such transparencies are not provided by any existing ODP-influenced system, e.g. OMG CORBA [CORBA] or Microsoft DCOM [DCOM].

4.2.3 The ODP Trading Function

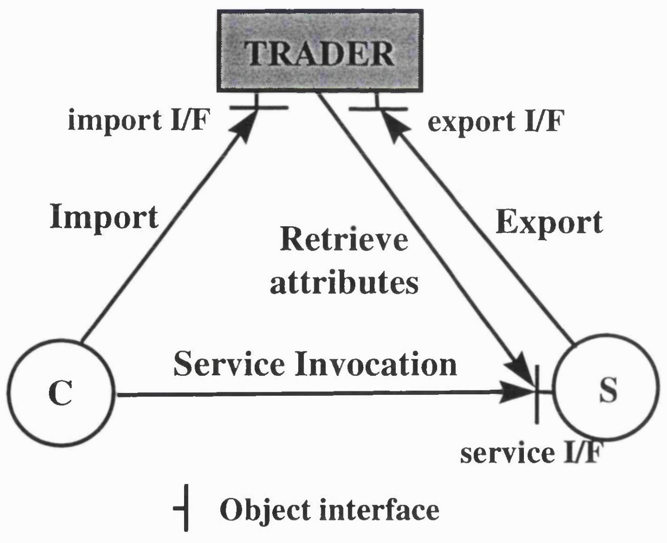


Figure 4-3 The ODP Trading Function

Objects "floating" on the ODP platform need to be able to locate each other before invoking distributed operations, which are supported by a server object's *interfaces*. Discovery of a server object is supported by the ODP trading function which is depicted in Figure 4-3. The ODP *Trader* [ODP][X9tr] is a special server that holds information about a "service space", manifested by the interfaces of various types of server objects. The latter *export* information to the trader about the interfaces they support and associated static *properties* or dynamic *attributes*. Properties are fixed for the lifetime of an object instance, e.g. the "next hop address" of a route object, while attributes may change dynamically, e.g. the current number and size of jobs in a printer queue.

Client objects wishing to contact a particular type of server interface contact the trader and pass a request that may include assertions on interface properties and/or attributes. The trader may need to retrieve the asserted attributes from all the objects with interfaces of the required type, evaluate the client's request and pass back the matching interface *references*. The function of the trader is in fact similar to the OSI-SM filtering function, but only test for equality is possible: this is because ODP interface attributes do not have "MATCHES FOR" properties in a similar fashion to GDMO attributes. Also from an engineering perspective, filtering is tightly coupled with the OSI-SM MOs while the ODP trader is a separate server object that may operate on a different node. Traders need to be federated in order to be able to cope with big service spaces and different administrative domains. Issues of trader federation have not yet been addressed in detail. The use of the OSI Directory [X500] as supporting technology is mentioned in the relevant recommendations since it supports federation.

Trading is generally important in distributed systems for locating server objects that satisfy particular criteria, e.g. locate the printer server with the smallest job queue. On the other hand, a subset of trading is mostly used in distributed environments, known as *name serving*. Manager or client objects need to be able to locate managed or server objects by name, given the fact that there will typically exist more than one instance of a particular type of managed object in the distributed environment. Though ODP standards do not explicitly mention name serving, associated technologies such as the OSF Distributed Computing Environment (DCE) [DCE] and OMG CORBA [CORBA] provide name serving facilities. A Name Server (NS) is a simplified form of a trader which only stores the name properties of server interfaces and resolves supplied names to interface references. The reader is reminded here that name serving in OSI-SM / TMN environments is supported by the OSI Directory, as described in section 2.4 of Chapter 2. In this case the server object is the application in agent role which exports its "global" name to the directory.

The trader depicted in Figure 4-3 exhibits two distinct interfaces: an export interface used by exporters or servers and an import interface used by importers or clients. In general, ODP objects may have more than one interfaces, bound to a common state through the object. In fact, ODP objects are characterised by the interfaces they support, which means that object interfaces, as opposed to objects themselves, are the "first class citizens" of the ODP "society". It should be noted that the only ODP-based technology that supports multiple distinct interfaces per object is the MS DCOM.

4.2.4 The ODP Viewpoints

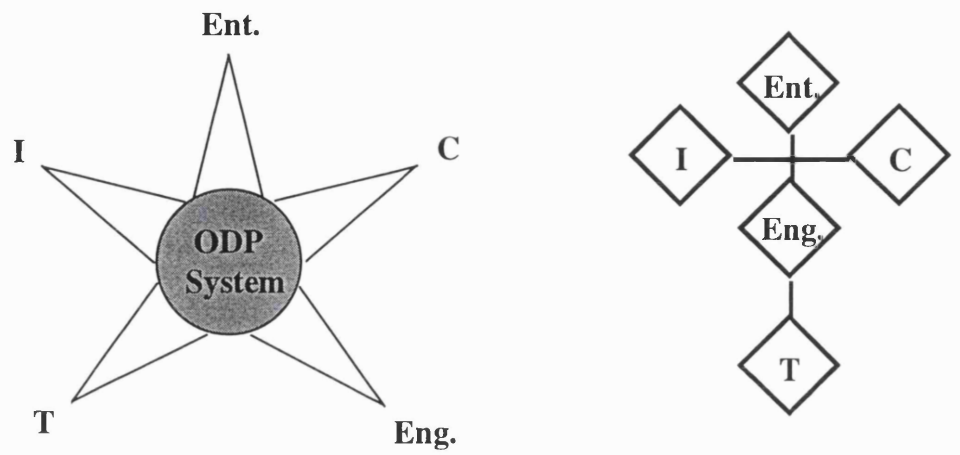


Figure 4-4 The ODP Viewpoints

Another important aspect of ODP is its specification methodology. The ODP prescriptive model [X903] proposes the use of five *viewpoints* which, when considered together, provide a complete framework for distributed system specification. A viewpoint is a form of abstraction of the overall system, focusing on particular concerns. The five ODP viewpoints are the following:

- the *Enterprise* viewpoint, which focuses on the business view of the system, including the purpose, scope, services offered, actors involved, their roles and business policies;
- the *Information* viewpoint, which focuses on the information content of the system, including the semantics and relationships of information objects; the information properties of a system constitute its information model;
- the *Computational* viewpoint, which comprises a description of the system in terms of interacting objects, focusing on the functional decomposition of the system for the purpose of distribution;
- the *Engineering* viewpoint, which focuses on the functions required to support distribution, including the provision of distribution transparencies by the ODP platform; and

- the *Technology* viewpoint, which focuses on real world artefacts from which the system is constructed i.e. hardware, software, operating systems, databases, etc.

Figure 4-4 depicts two views of the five viewpoints. The left view emphasises the fact that each viewpoint projects a different view of the same system. The right view shows a hierarchical relationship of the viewpoints: the enterprise view comes first, representing the “raison d' être” of the system; the system design follows through the information and computational decomposition; the system is subsequently deployed in the engineering viewpoint; finally, the technology viewpoint documents the technology aspects.

Each viewpoint may have one or more associated viewpoint languages that define the concepts and rules for representing the system from that viewpoint. Natural or formal, textual or graphical notations may be used as required by that particular viewpoint. An important issue is that the overall specification of the system from the five viewpoints should be consistent.

The aspects of the enterprise viewpoint are typically documented in natural language. In the case of the TMN, the description of the management services [M3200] and the relationships between the administration operating the TMN and other administrations or human users can be thought as part of the enterprise viewpoint.

The information and computational viewpoints are intimately coupled, since they both express the design of the system. On the other hand, it is the computational viewpoint that defines the distributed interactions between objects and the structure of messages exchanged. In the simplest case, there is a one-to-one mapping between information and computational objects. In other cases, a computational object may hold information objects internally, providing access to them through its computational interface. The relationship between the ODP information and computational viewpoints is generally a very complex issue and not particularly well addressed. The TINA information modelling concept document [Crist95] includes an interesting discussion on this relationship, concluding that no general rules can be drawn. We will investigate in detail this separation in the context of OSI-SM / TMN later in this chapter.

The OSI-SM GDMO [X722] and GRM [X725], despite the fact they were developed for the purpose of defining managed objects and their relationships, represent formal techniques for general information modelling. A slightly modified version of those known as Quasi-GDMO/GRM is used as an information modelling language in TINA [Crist95]. Another widely-used notational technique for describing information models in the ODP information viewpoint is the Object Modelling Technique (OMT) [Rumb91]. This has also been used throughout this thesis for describing static class relationships (see Appendix B).

Object interfaces are specified in the computational viewpoint using an Interface Definition Language (IDL). The methods of the relevant interfaces and their arguments define the messages that can be sent to the objects that support those interfaces. Two types of interfaces can be supported by ODP objects: *operational* interfaces which support request-response operations; and *stream* interfaces, which support continuous information flows, e.g. for audio and video streams, file transfer, etc. Formal languages such as SDL [Z100], Z [Spiv89] may be used in addition to GDMO/GRM and IDL in the information and computational viewpoints in order to specify object behaviour.

The engineering viewpoint maps the computational objects to engineering objects supported by the ODP platform, introduces deployment concepts and supports the distribution transparencies. Computational objects are mapped to Basic Engineering Objects (BEOs). A remote BEO is represented to a local BEO through a *stub* object, which provides access transparency through functions known as *marshalling* and *unmarshalling*; these are equivalent to OSI presentation layer functions. A *binder* object maintains the binding between BEOs and can be thought as analogous to an OSI association between application layer entities. Finally, the stub and binder objects are supported by a *protocol* object which encapsulates the underlying protocol stack. Related BEOs may be grouped together in *clusters*, which become the unit of distribution i.e. they can be deployed and migrated together. Clusters may be grouped together in *capsules*, which typically map onto operating system processes. Capsules are deployed in *nodes*, which represent computing equipment attached to the network e.g. a general purpose workstation, a piece of switching equipment, etc. Finally, nodes contain the *nucleus* which realises the basic ODP platform environment.

4.2.5 OSI-SM / TMN and the ODP Viewpoints

Since the early days of the ODP framework, there have been efforts trying to describe OSI-SM in ODP terms. [Proct92] provides an analysis of OSI-SM from an ODP perspective, parts of which were later added as an example to the ODP recommendations. More recently, the Open Distributed Management Architecture (ODMA) [X703] does the same but in a more detailed manner and from the point of view of the OSI-SM recommendations. Various other researchers have addressed the issue of OSI-SM in the context of ODP. While we are going to describe relevant research and present our view in detail later on in this chapter, we consider briefly the relevant issues below.

There are two basic approaches for describing OSI-SM / TMN in ODP terms, which can be thought as being at the two ends of the relevant spectrum:

- every OSI-SM managed object becomes an ODP computational object, with manager objects interacting directly with them; or
- every OSI-SM / TMN agent becomes an ODP computational object, providing access to the contained managed objects which can be thought as ODP information objects mapped directly to engineering objects.

In the first case, the computational interface of every managed object is derived by mapping its GDMO specification to IDL. In this case, ODP mechanisms are used for object discovery, e.g. name servers and traders, and event dissemination, e.g. ODP event servers. In the second case, the computational interface of the agent is produced by mapping the CMIS specification to IDL. In this case, the OSI-SM mechanisms are used for object discovery i.e. scoping / filtering, and event dissemination i.e. event forwarding discriminators. In both cases, the communications mechanism can be either the Q_3 protocol stack, assuming that the ODP platform is ported over CMIS/P, or the protocol stack prescribed by ODP.

The OSI-SM/TMN environment described in Chapter 3, though not conforming to the letter of the ODP recommendations, can be thought as a distributed management software platform in the "spirit" of ODP. It actually uses the second approach described above but includes aspects of the first approach as follows. The distributed interactions follow the second approach, with manager-agent exchanges take place across Q_3 interfaces for TMN compliance. On the other hand, manager and managed objects are presented with APIs similar to those of ODP-influenced platforms. Manager objects can access either individual managed objects through the SMIB API, in a similar fashion to the first approach above, or whole agents through the RMIB API, in a similar fashion to the second approach. In fact, information objects are accessed through APIs as if they were separate computational objects. We will investigate the issues of mapping the OSI-SM / TMN model onto ODP-influenced distributed object frameworks in more detail in the rest of this chapter.

We will finalise this introduction to ODP with a comment on its use as a methodology for designing and specifying distributed systems. While the ODP viewpoints present an excellent set of tools for documenting the design of a complex distributed system, they are just that - a set of documentation tools. There is no clear way to go from one viewpoint to another, and the relationships between the information and computational views are not fully addressed. The word methodology implies a well-defined approach for system decomposition and component identification. A methodology for TMN interface specification was discussed in section 2.3.2.4 of Chapter 2 [M3020][Gri96a], explaining how the designer starts from management service definition, moves into functional decomposition and then recomposition in order to arrive at a computational view. Such aspects are currently lacking in ODP. In fact, some refer to ODP as a *meta-methodology*, which needs to be specialised for a particular problem domain.

4.3 ODP-based Technologies

In this section we consider the Advanced Networked Systems Architecture (ANSA), the OSF Distributed Computing and Management Environment (DCE/DME) and the OMG Common Object Request Broker Architecture (CORBA) as ODP-based technologies. We concentrate more in OMG CORBA, since it will be considered as the base technology for the TMN in section 4.4. This presentation serves as a state-of-the-art description but also goes further, discussing issues behind the use of those technologies in TMN environments. In addition, the presentation of OMG CORBA addresses aspects that are not typically discussed in the relevant literature.

4.3.1 The Advanced Networked Systems Architecture

The Advanced Networked Systems Architecture (ANSA) [ANSA89a] was the first distributed software platform. It took further the concept of RPC-based software infrastructures compared to previous efforts by Sun Microsystems [Sun87][Sun88] and UCL [Wilb87]. ANSA was more than a distributed software platform, influencing the development of the whole ODP framework. In this section, we examine ANSA mainly as a distributed software platform and consider its suitability for TMN environments.

The ANSA platform, known as ANSAware, uses its own remote procedure call protocol. Object interfaces are specified in the ANSA IDL. Multiple objects can be organised into capsules, which map onto operating system processes. Objects in different nodes interact through the Remote Execution Protocol (REX), which relies on the underlying RPC protocol. Both synchronous and asynchronous remote execution mechanisms are supported. Remote operations are invoked through pre-processor extensions to the underlying programming language, which are specified in the Distributed Processing Language (DPL). The ANSAware environment and application APIs are in the C programming language.

Let's examine now ANSA against the desired properties of object-oriented distributed frameworks defined in section 3.2.2 of Chapter 3. Though ANSA claims to be object-oriented, it is in fact object-based. Its IDL does not support inheritance and polymorphism while the relevant APIs are in the C programming language. Though Modula 3 and C++ language mappings were promised, they were never officially released. Remote operations are invoked by clients in a "pseudo" object-oriented fashion, through DPL directives i.e. real proxy objects do not exist in the local address space in a similar fashion to CORBA or to the OSIMIS RMIB/SMIB. Finally, there is no dynamic invocation interface.

The author considered porting the OSIMIS environment over ANSA during the first year of the RACE ICM project i.e. in 1992. The main reason was the increasing interest in ODP and relevant technologies in European telecommunications research projects. In addition, OSIMIS did not support at the time location transparency, so the ANSA trader could be used for this purpose.

This mapping could follow one of the two approaches described in section 4.2:

- every managed object would become an ANSA object, with its own IDL interface resulting from the GDMO to ANSA IDL mapping; or
- the whole agent would become an ANSA object, with an IDL interface mirroring the functionality of CMIS.

The first approach poses the problem that GDMO is object-oriented, supporting inheritance and polymorphism, while the ANSA IDL is not. It is of course possible to turn an inheritance hierarchy into a flat structure by repeating the inherited functionality. Such an approach though is a "hack" and does not support reusability and extensibility. In addition, if ANSA was adopted event dissemination facilities would need to be provided. With the OSIMIS infrastructure in place, including scoping, filtering, fine grain event reporting, logging and high-level object-oriented APIs in C++, this approach seemed to be more than one step backwards and was rejected.

The second approach means that the OSIMIS generic agent and manager infrastructures could be retained, including name resolution, scoping, filtering and event dissemination. The whole agent would become an ANSA object, with the manager-agent communications based on some form of "Q_3" interface over the ANSA protocols. Management applications would discover each other through the ANSA trader.

This approach seemed more promising and the author started investigating it but he soon faced a brick wall. Realising the CMIS service in IDL requires support for an *any* type, with semantics similar to the ASN.1 ANY, and the ANSA IDL did not support such a type. In addition, some CMIS operation parameters, e.g. filters, are "recursive" ASN.1 structures which cannot be expressed in ANSA IDL. CMIS is by definition an asynchronous service. While ANSA claimed to support asynchronous operation invocations through a "voucher" mechanism, the latter did not seem to work in the ANSA version at the time. ANSA provided a threads mechanism to be used with synchronous operations. This operated in user space and made distributed programs difficult, if not impossible to debug. As such, this approach was never pursued any further. A variation of this approach was pursued though by the author a few years later using CORBA, the latter being the first ODP-based technology that had come of age. The CORBA-based approach will be described in detail in section 4.4.

It should be clear from the above discussion that ANSA represented a fairly early view of an ODP-based infrastructure and had a number of limitations. Due to those limitations, it is not possible to use ANSA as base technology for TMN systems. On the other hand, ANSA paved the way to more mature technologies such as CORBA and provided both the theoretical foundation and a first concrete implementation as a proof of concept for the ODP framework.

We will close this discussion on ANSA with a final remark. While ANSA claimed to be an *open* architecture, providing a solution for *open* distributed processing, this claimed openness is debatable. ANSA used its *own* protocols, despite the fact there was a lot of talk about substituting those with agreed protocols. In addition, there was no attempt to standardise the relevant APIs. There has always been only *one* ANSA product by Architecture Projects Management (APM) Ltd, the company formed to develop and promote ANSA. ANSA applications can only interwork with other ANSA applications on the same platform. All these aspects point to a *closed* rather than *open* solution. The author expressed those views at the time. [Marc95] makes exactly the same remarks for the OSF DCE/DME, which we will examine in the next section.

4.3.2 The OSF Distributed Computing and Management Environment

The Open Software Foundation (OSF) Distributed Computing Environment (DCE) [DCE] is an integrated collection of a support environment, tools and services for the development, deployment and maintenance of distributed applications. Its core components include the DCE RPC protocol and programming environment; a threads facility; the DCE Directory Services, including the Cell Directory Service (CDS) used for naming and location transparency; the DCE Security Service which provides Kerberos-based authentication, stream integrity and access control; and the DCE Distributed Time Service (DTS). The idea of the DCE was to provide a vendor-neutral platform for building distributed applications, building on work done by various OSF vendor members and reusing their software infrastructure. In this section we consider briefly DCE as an a potential environment for distributed TMN applications.

DCE object interfaces are specified in the DCE IDL, which is different to the ANSA IDL. Multiple objects can be organised into DCE *servers*, which map onto operating system processes. Communication between those servers is supported by the DCE RPC protocol. The DCE IDL does not support inheritance and polymorphism, while the relevant APIs are in the C programming language. DCE is a procedural environment instead of being object-based or object-oriented. Clients call directly procedures of interfaces, without a proxy object in the CORBA/OSIMIS sense, or even a "pseudo" object handle, as in the ANSA DPL. In other words, DCE is simply an RPC environment, with additional support services for naming and security. It would be rather stretched to call it an ODP-based environment. The reason we consider it here is that it was a commercial attempt towards a distributed software platform that was fairly popular before the advent of fully distributed object technologies such as OMG CORBA and MS DCOM.

The same issues regarding the use of ANSA in TMN environments hold also for DCE. It is not possible to map directly the OSI-SM/TMN model onto DCE using any of the two approaches described above: the problems described for the ANSA IDL hold also for the DCE IDL. The reader may remark that it is possible to use a basic mechanism, such as the DCE IDL and RPC, and build on top of them a fully object-oriented distributed framework. For example, MS DCOM uses DCE while [Autr94] describes how to map the CORBA IDL onto the DCE IDL. While this is possible, we examine here the possibility of using DCE *as is* rather than using it as a starting point for developing a whole new framework. In summary, DCE is *not* suitable technology for TMN systems.

While DCE targeted general purpose distributed systems, an OSF initiative known as the Distributed Management Environment (DME) [DME] tried to build on it and provide a distributed object-oriented management framework. DME tried to combine every known management technology into one integrated framework. It uses DCE for the communications between management applications but it also uses Q_3 and SNMP for accessing managed elements with such interfaces. While the DCE is hidden underneath the DME, applications are built using the CORBA framework and APIs which are in turn mapped to the DCE IDL and RPC. Adapter objects were supposed to convert between the IDL/RPC framework and CMIS/P or SNMP. A fine grain event service would be provided to applications while the software platform would include a generic graphical user interface, similar to a MIB browser [Pav92a].

The DME idea originated from the fact that DCE provided an interoperable communications environment with naming services and security, while CORBA provided the object-oriented APIs but not an interoperable protocol at the time. NMF and X/Open had just started their Joint Inter-Domain Management (JIDM) task force to investigate the mapping of GDMO and the SNMP SMI to CORBA IDL, so it was thought easy to provide adapters to Q_3 and SNMP. The idea was to reuse existing software components from Hewlett-Packard, IBM and Tivoli. It is not surprising that the DME "dream" never materialised. A discussion on the reasons behind DME's failure can be found in [Marc95], which is titled "Icaros, Alice and the OSF DME". The title alludes that DME tried to fly "too close to the sun", as Icaros in ancient Greek mythology. It also points to Lewis Carroll's Alice in Wonderland!

While the DME approach seemed unrealistic at the time, relevant research since then has made possible the combination of protocol-based TMN solutions and new generations of distributed object technologies such as CORBA and DCOM. Related work and the author's approach will be presented in section 4.4.

4.3.3 The OMG Common Object Request Broker Architecture

4.3.3.1 The Object Request Broker Model

While the ISO/ITU-T ODP [ODP] is a theoretical framework for specifying and building distributed systems, the Common Object Request Broker Architecture (CORBA) [CORBA] can be seen as its pragmatic counterpart. The relevant architecture and specifications have been the result of work by the Object Management Group (OMG) and represent an industrial approach towards open distributed systems. A number of vendors offer CORBA platforms that conform to the relevant specifications. These provide both "horizontal" openness through standard APIs, resulting in application portability across platforms, and "vertical" openness through agreed protocols, resulting in application interoperability across platforms.

The OMG CORBA model is shown in Figure 4-5. The relevant paradigm is a client-server one, with distribution transparencies provided through the Object Request Broker (ORB), which has similar functionality to the ODP platform. The unit of distribution is the single object as opposed to the OSI-SM object cluster. Objects in client and server roles communicate through the ORB. Special server objects provide generic Common Object Services [COSS], such as naming, event serving, trading, etc. Interoperability is achieved through the formal specification of server interfaces in the CORBA Interface Definition Language (IDL) and through the underlying protocols. Portability is achieved through standard APIs for objects in client and server roles. These map the CORBA IDL onto various programming languages, such as C, C++, Smalltalk, Java, etc. The CORBA object model, i.e. the IDL language which defines the properties of interfaces and subsequently objects, and the relevant APIs have been addressed first, while the underlying protocols may be replaced.

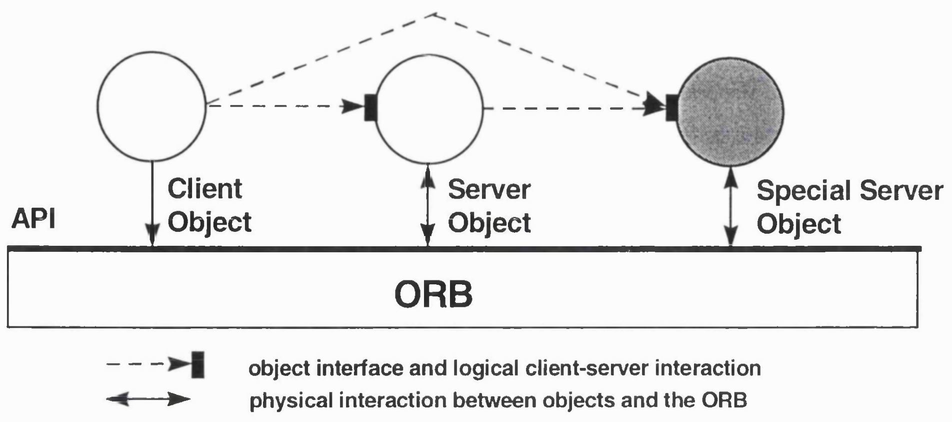

API
Client Object
Server Object
Special Server Object
ORB
object interface and logical client-server interaction
physical interaction between objects and the ORB

Figure 4-5 The OMG CORBA Model

CORBA objects can be organised into "containers" similar to the ANSA/ODP capsules, which are called *servers* and map onto operating system processes. Note that this use of the term server is different to the same term referring to an object in server role: CORBA objects in client and server roles may be organised into servers i.e. operating system processes. Objects within a server may communicate either directly, exploiting the knowledge they are co-located within that server, or indirectly through the ORB. A CORBA server is in fact equivalent to both the ODP concepts of capsule and cluster. In other words, there is no notion of cluster in CORBA but only the notion of capsule. The ORB itself is a special daemon that runs in every CORBA-capable network node. Note that the ORB as depicted in Figure 4-5 models the union of all those ORB daemons running on different network nodes. This conceptual distributed ORB together with the various generic servers that offer special services is also referred to as the Distributed Processing Environment (DPE); the latter is terminology originated in TINA [TINA].

Since CORBA is an object-oriented, mature, ODP-influenced technology, it can be considered as the basis of TMN systems in lieu of the OSI-SM and Directory. As such, we will consider it here at the same level of detail as we did for OSI-SM in section 2.2 of Chapter 2, covering both its information modelling and distributed object access aspects.

It should be noted that OMG CORBA is not the only emerging distributed object technology: Microsoft's Distributed Component Object Model (DCOM) [DCOM] is another similar technology. The key difference between CORBA and DCOM is that the former is based on collectively produced, openly available specifications by the OMG, while the latter is the work of a single company. Any vendor may develop a CORBA-compliant platform and compete in the relevant marketplace while there exists only one DCOM platform, produced by Microsoft. This is one of the key reasons we consider CORBA instead of DCOM in this thesis. Despite that, and given the similarities between CORBA and DCOM, we will refer specifically to any differences between the two during the detailed presentation of CORBA.

4.3.3.2 The Information Model

The CORBA information model is "indirectly" defined by the Interface Definition Language (IDL), which defines the properties of object interfaces and subsequently objects. In the strict ODP sense, IDL is a computational viewpoint language since it defines the interactions among objects through their interfaces. On the other hand, in the absence of any formal information modelling language that is specific to CORBA, we can also treat IDL as an information modelling language.

The CORBA information model is fully object-oriented, in a similar fashion to OSI-SM. Objects are characterised by the *interfaces* they support. While an ODP object may support multiple interfaces bound to a common state, the current OMG IDL allows only a single interface per object. It should be noted here that the DCOM IDL allows multiple interfaces per object. In fact, the OMG model defines objects through the specification of the relevant interfaces. As such, there is no direct concept of an object class. Object interfaces may be specialised through inheritance while multiple inheritance is also allowed. The root interface in the inheritance hierarchy is of type *Object*. The IDL specification technique is more monolithic than the GDMO piecemeal approach: the minimum re-usable specification entity is the interface definition as opposed to the individual package, attribute, action and notification in GDMO. IDL may be regarded as broadly equivalent to the GDMO/ASN.1 combination in OSI management, though less powerful and with some important differences highlighted below.

An OMG object may accept operations at the object boundary, have attributes and exhibit behaviour. Such an object is used to implement a computational construct. In a management context, an object may behave as a manageable entity, modelling an underlying resource.

Objects accept object-oriented operations, similar to the GDMO actions. Exceptions may be defined as part of an interface and be associated to operations. The latter take *in* and *out* arguments, which may be of arbitrary IDL types. Objects may also have attributes of arbitrary IDL types. Attributes accept *get* and *set* operations while those specified as *readonly* accept only *get* operations. Attributes can not be specified with "set by default", "add" or "remove" properties as in GDMO. In addition, only standard exceptions may signify an error during the get and set operations. This is in contrast to GDMO, where arbitrary class-specific errors may be defined to model exceptions for attribute operations. Defining an attribute as part of an object interface results simply in methods named *<attr>_get* and *<attr>_set* to be part of that interface. The limitations discussed above may be lifted if, instead of defining attributes, an information model designer defines directly associated access methods. In this case, *<attr>_setToDefault*, *<attr>_add* and *<attr>_remove* methods could also be defined, while all the attribute access methods may have associated exceptions.

A key difference between OMG and GDMO objects is that the former do not allow for the late binding of functionality to interfaces through optional constructs similar to the GDMO conditional packages. An OMG object type is an absolute indication of the characteristics of an instance of that type. An additional major difference is that in IDL it is not possible to specify event types *generated* by an object: events are modelled as "operations in the opposite direction". As such, events are specified through operations on the interface of the receiving object. An OMG managed object needs to specify a separate interface containing all the events it can generate; the latter needs to be supported by managing objects that want to receive those events. This peer-to-peer modelling of events is shown in Figure 4-6. The interface A models the operations supported by object A while the interface B models the events emitted by object A and received by object B. This split in the specification of operations and events is one key drawback of the model. There are more differences with respect to the way events are disseminated, but these are discussed in the next section, since they are related to the access paradigm.

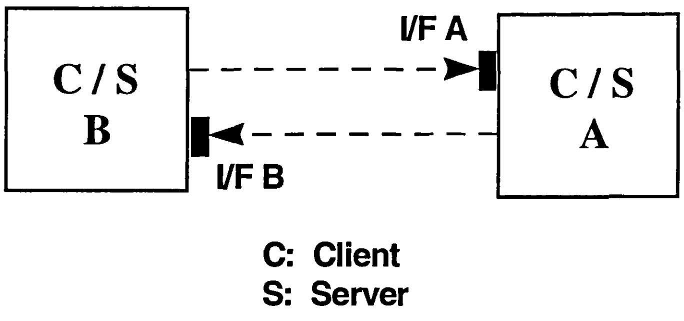


Figure 4-6 Peer-to-Peer Object Interaction for Events and Asynchronous Operations

OMG objects do not provide a built-in operation for instantiation of interfaces by client or managing objects. The reason for that is that OMG takes a "programmatic" view of object interfaces and, as such, a create operation is meaningless before that interface exists. While GDMO objects appear to accept create operations, these are essentially targeted to the agent administering them. As such, interface creation in OMG may only be supported by existing interfaces: *factory* objects may be defined that allow client objects to create application specific interfaces. This approach is not flexible since a specific factory interface is necessary for every interface that can be dynamically created.

Deletion of objects is possible through the OMG Object Life-Cycle Services [COSS]. The latter has specified an interface that provides a *delete* as well as *copy/move* operations. Any other interface that needs to be deleted should inherit from the life-cycle interface. The copy/move operations apply to object implementations and appear to be very powerful as they support relocation and replication. The downside is that they have not yet been supported in practice. In the absence of implementations supporting life-cycle services, interface deletion is currently tackled through the definition of interface-specific *delete* operations. The problem is that if an object receives a delete request through its interface and deletes itself, there is no reply to the performing client. An exception is raised instead and the client will never know if deletion was completed successfully or something else went wrong while the object was cleaning-up its state. Hopefully mature implementations of the life-cycle service interface will solve such problems in the future.

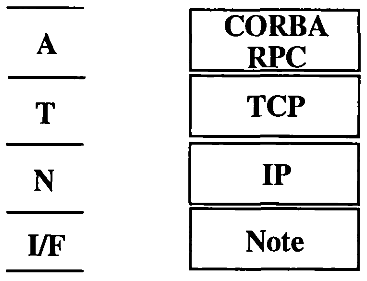
While the OMG IDL object model has many similarities to GDMO, a marked difference concerns naming. OMG objects can be identified and accessed through *Object References*. The latter are assigned to objects at creation time and are opaque types, i.e. they have no internal structure and, as such, do not reveal any information about the object. Object references are in fact similar to programming language pointers and are used to de-reference a "proxy" server object within a client's address space. While distributed objects are effectively accessed through references, an object may also be assigned one or more names. The latter are distinct from objects, unlike OSI-SM where an object always has a name. Actually OMG objects need not have names at all as they may accessed through their interface reference. In addition, names may be assigned to objects but this mapping may change at any time. Names are assigned to objects through the Name Service [COSS], which provides a directed graph of naming contexts with potentially many roots. A point in the graph may be reached via many routes, which means that an object may have many names. This is in contrast to OSI-SM where there exists a naming tree instead of a naming graph and objects have exactly one name. The name service may be used to assign names to objects and to resolve names to object references.

Finally, while the CORBA IDL supports inheritance, it does not support polymorphism in the same fashion as in object-oriented programming languages or in GDMO: it is *not* possible to redefine an operation or attribute in a derived interface and invoke its behaviour through a reference to the parent interface while this is possible in GDMO. This is a particularly important limitation which shows that inheritance in CORBA IDL is simply a means of "bundling" properties together rather than using it for designing generic, polymorphic distributed systems. In a related fashion, there is no support for allomorphism.

4.3.3.3 The Access Paradigm

OMG CORBA was designed as a distributed software infrastructure in which the access protocol is secondary compared to the underlying APIs, which represent agreed IDL mappings to programming languages. Of course, an agreed protocol is necessary in order to achieve interoperability between CORBA platforms of different vendors. The OMG 1.x versions of CORBA specification left completely open the choice of access protocol and concentrated only on programming language bindings. Version 2.0 specifies a Remote Procedure Call (RPC) protocol, known as the General Inter-Operability Protocol (GIOP) [GIOP]. Two different transport mappings have been currently defined for the latter, the Internet Inter-Operability Protocol (IIOP) [IIOP] over the Internet TCP/IP (shown in Figure 4-7) and the DCE Common Inter-Operability Protocol (D-CIOP). IIOP has been by far the most widely deployed open protocol suite, in fact few commercial CORBA platforms support the D-CIOP. The rest of the CORBA software infrastructure is relatively independent of the underlying protocol, which could change without any effect on the APIs and the application software.

The GIOP/IIOP is a connection-oriented reliable RPC protocol that uses TCP and IP as transport and network protocols respectively. Applications that use CORBA-based communications are guaranteed transport reliability, in a similar fashion to OSI-SM. The RPC protocol is a general *request/response* protocol in which the exact content of the request and response packets is governed by the IDL specification of the accessed CORBA interface, in a similar fashion to the CMIS/P and GDMO coupling. The access service / protocol offers no special facilities for object discovery, similar to the OSI-SM scoping and filtering. Such facilities are provided instead by naming servers and traders. In summary, the CORBA RPC protocol provides a *single* object access mechanism, with higher-level facilities provided by standard OMG servers [COSS].



Note: The Internet interface layer may comprise many different protocols

Figure 4-7 The CORBA IIOP Protocol Stack

Objects in client roles interact with objects in server roles through "proxy" objects available in the local address space. An object reference is required in order to be able to get access to the proxy object representing a remote object in server role and invoke its methods. This reference can be obtained through the ORB by "binding" to the remote object by type, if the host where the object is running is known. The problem with this approach is that it violates location transparency. The correct approach which supports location transparency is to discover an object's reference either by name, through the name server, or by type and potentially other properties through the trader. A naming server [COSS] can be used to map one or more names to an interface reference. When an object instance is created, the naming server needs to be "notified" of the mapping between the object's name and its interface reference. Subsequently, client objects can resolve object names to object references through the naming server. A trader [X9tr] supports more sophisticated queries, matching static properties and dynamic attribute values of the target object(s), as it was described in section 4.2.3.

Having obtained a reference to an object, operations are invoked through the associated proxy object in a *synchronous* fashion only. The only possibility to perform operations asynchronously is to define them as *oneway* in the IDL specification. "One way" operations cannot have associated output arguments and exceptions. When defining one way operations, a designer needs to define as well the symmetric operations which model the relevant results and errors; these will be part of the client's interface, with the client and server roles reversed for the forwarding of the results / errors. The object reference of the client is typically passed as an argument in the "master" one way operation so that the server can "return" the results / errors. The peer-to-peer interaction for asynchronous operations was depicted in Figure 4-6. The interface A is "half" of what it should normally be, while the interface B is the other "half" of A, modelling the results and errors for the operations to A.

This approach to asynchronous operations is hardly satisfactory. Operations need to be defined as both oneway and non-oneway, with the results / errors requiring a separate interface. This increases considerably the complexity for asynchronous operations. In addition, the whole approach is somehow a "hack": asynchronicity should be a property of the API rather than the IDL interface via the oneway mechanism. For example, the OSIMIS RMIB/SMIB APIs described in the section 3.4 of Chapter 3 support both synchronous and asynchronous modes for invoking operations on remote GDMO objects. Given the significance of an asynchronous mode of execution in single-threaded environments, OMG intends to address asynchronous operations properly in CORBA version 3.0 [Vino97].

Operations invoked on proxy objects are strongly-typed and require the existence of static, precompiled knowledge of the target IDL interface in the client's program, in a similar fashion to the static OSIMIS SMIB model. CORBA supports also a Dynamic Invocation Interface (DII) which overcomes this limitation. The DII is a dynamic, weakly-typed interface that uses a generic operation interface. Operation types are specified in string form while the arguments, results and exception parameters are specified as IDL *any* types. This is similar to the OSIMIS RMIB and dynamic SMIB models. The DII supports also an asynchronous mode of operation which does not require operations to be defined as one way operations; this is in fact known as the *deferred synchronous operation* model. The DII could be used for generic applications such as MIB browsers etc. It should be noted though that not all commercial CORBA platforms support the DII. In fact, the author has not experienced any use of the DII in the various European research projects which have been using CORBA.

Finally, notifications in ODP/OMG are supported by *event* channels. Emitting and recipient objects need to register with those channels which are created and managed for every type of notification. Emitting objects invoke an operation on the relevant event channel while the notification is passed to registered recipient objects either by invoking operations on them (*push* model) or through an operation invoked by the recipient object (*pull* model). The interaction depicted in Figure 4-6 reflects the push model, while the event channel object is not shown. Event channels can be in principle federated for scalability and inter-domain operation but the relevant issues have not yet been addressed. The key difference with OSI-SM is the lack of fine grain filtering, which results in less power and expressiveness and more management traffic. OMG is currently working towards the specification of *notification* servers which will provide filtering and will also undertake the management of channels, providing a higher-level way to deal with notifications.

4.3.3.4 Summary

It should be clear from the above presentation that though CORBA is not as powerful as OSI-SM, it is in an ODP-influenced technology that has come of age and has a number of the attributes sought for supporting distributed network and service management systems. We will finalise this section by examining CORBA against the desired properties of object-oriented distributed frameworks, as defined in section 3.1.2 of Chapter 3.

CORBA projects a fully object-oriented framework since its IDL supports inheritance while there exist standard mappings to object-oriented programming languages. Its APIs are user-friendly, hiding communication and protocol details through "proxy" objects that represent remote objects as if they were in the same address space. They are in fact similar to the OSIMIS SMIB [Pav94b] and the NMF GDMO/C++ [Chat97] models that were discussed in Chapter 3, though they do not support information caching mechanisms. CORBA supports a Dynamic Invocation Interface (DII) which does not require pre-compiled knowledge of remote IDL interfaces in clients, providing for a dynamic, weakly-typed style of interaction suitable for generic applications. CORBA is open both in terms of standard APIs and communications protocols. Finally, CORBA supports access transparency through the IDL and the ORB APIs. It also supports location transparency through the naming server and trader, which are special servers.

4.3.4 A Summary of the Relevant Technologies

Having presented ANSA, the OSF DCE and OMG CORBA as ODP-based technologies, we will finalise this section with a summary and comparison of the characteristics of those technologies in relation to the desired properties of object-oriented distributed frameworks. We will also add as a fourth technology the combination of OSI-SM together with its realisation through the OSIMIS platform. Though OSI-SM / OSIMIS are not, strictly speaking, ODP-based, we have shown in this thesis that they exhibit a number of aspects of ODP systems and, as such, we can consider them in this summary and comparison.

It should be emphasised again that OSI-SM is a compositional model, based on the "object cluster" principle, so it cannot be used instead of any of those other technologies (or rather it can, but this is a degenerate case). On the other hand, any of the other technologies could be considered instead of OSI-SM. Table 4-1 shows the relevant technologies and their properties which are summarised below.

ANSA's IDL is not object-oriented while remote objects are accessed through DPL pre-processor extensions, without true proxy objects. There have not been mappings to C++, Smalltalk or Java but there has been an experimental mapping to Modula 3. DCE's IDL is not object-oriented, the APIs are procedural while there exists only a mapping to C. Both ANSA and DCE do not support a dynamic invocation interface and the *any* type in their IDL.

CORBA's IDL is object-oriented but it does not support true polymorphism through the redefinition of operations and attributes in derived interfaces. The APIs are fully object-oriented and there exist multiple language mappings. A dynamic invocation interface and the any type are also supported.

OSI-SM's GDMO is fully object-oriented and supports inheritance and polymorphism. The OSIMIS APIs are fully object-oriented and support both static and dynamic invocation paradigms, in the latter case through the mapping of the ANY DEFINED BY ASN.1 construct to C++. A manager/client mapping to Tcl has been presented while there exist similar Java mappings in other TMN platforms. On the other hand, the major mapping both in OSIMIS and in the NMF TMN/C++ APIs is in C++.

Technology and properties	O-O Abstract Language	O-O Language Mapping	Multiple Language Mappings	Dynamic Invocation Interface
ANSA	no	no	partly	no
DCE	no	no	no	no
CORBA	yes/partly	yes	yes	yes
OSI-SM / OSIMIS	yes	yes	partly	yes

Table 4-1 A Comparison of Distributed Object Technologies

It is evident from the Table 4-1 that ANSA and DCE fail to satisfy important properties of object-oriented distributed frameworks. As such, they cannot be used as base technologies for the TMN for reasons discussed in detail in sections 4.3.1 and 4.3.2. It is also evident from the Table 4-1 that CORBA is an ODP-influenced technology that has come of age, since it satisfies almost all those properties. It lacks though full support for polymorphism, proper support for asynchronous APIs and event-reporting based on filtering.

4.4 Using Distributed Object Technologies in TMN

According to the analysis in the previous section, OMG CORBA is the first ODP-influenced technology that satisfies most of the requirements for building object-oriented distributed systems with dynamic properties. In this section, we will examine in detail the issues behind using CORBA and ODP principles in TMN environments. The key issue is to investigate exactly how OSI-SM can be replaced by CORBA as the base technology for the TMN. We need thus to investigate in detail the relationship between OSI-SM and CORBA or, more generally, the relationship between OSI-SM and ODP.

4.4.1 Mapping OSI-SM to ODP

In section 4.2.5, we discussed in a "flash forward" fashion two approaches for describing the OSI-SM model in ODP terms. In the first one, every managed object becomes a separate ODP computational object, with no OSI agent functionality. In the second one, every agent becomes an ODP computational object, containing managed objects as information objects. These approaches can be thought as being at the ends of the spectrum, while mixed approaches are also possible. The first approach has its origins in the ODP community while the second one was conceived by the author and other researchers trying to preserve the benefits of the OSI-SM model in ODP environments. We present the first approach below.

Since the early days of ODP, there have been various attempts to describe OSI-SM in ODP terms. [Proct92] was the first attempt, part of which found its way into the ODP-RM [ODP] as an appendix. The approach taken was to consider managed and managing objects as ODP objects, communicating via supporting ODP platform mechanisms. This implies that the functionality of OSI agents, such as name resolution, scoping, filtering, object creation and access control, is not explicitly present. The functionality of mapping object names to interface references and object creation need to be supported through ODP mechanisms. [Proct92] does not address those aspects but suggests that OSI-SM mechanisms should be used for event reporting through EFD support managed objects.

A similar approach has been more recently studied in the Open Distributed Management Architecture (ODMA) [X703], which is the ISO/ITU-T approach to describe OSI-SM in ODP terms. ODMA tries to provide a more general framework that can be mapped to either ODP-based object technologies, such as OMG CORBA, or to OSI-SM communication protocols and relevant engineering concepts. OSI managed and managing objects map onto ODP objects and interfaces. The ODP trader is used for discovering interface references, according to desired object properties. Object creation is supported through factory interfaces, which can be also discovered through the trader.

In the case of an ODP-based supporting technology, managing and managed objects communicate directly with each other over the ODP platform. When the underlying supporting technology is OSI-SM-based, the OSI agent becomes an "operation dispatcher" in the engineering viewpoint which performs operations to managed objects. It also becomes a "notification dispatcher" that forwards notifications to managing systems. One or more notification dispatchers may be also necessary in a managing system in order to deliver the notifications to the requesting managing objects. This is *exactly* the functionality provided by the MIBAgent object in OSIMIS agent applications and by the RMIBAgent objects in manager applications. This means that the object-oriented decomposition presented in Chapter 3 is in accordance with the engineering decomposition of OSI-SM systems in ODP terms, as suggested in [X703].

This ODP view of OSI-SM implies that the resulting framework does not support scoping, filtering and multiple operations to managed objects. In addition, if CORBA is used as the underlying platform, it is mentioned that notifications should be disseminated using relevant mechanisms i.e. OMG event servers and channels. When OSI-SM based platforms are used, the relevant protocols and supporting engineering concepts such as agents and notification dispatchers should be hidden behind the ODP platform APIs. The intention is to allow for the specification of management systems from an information and computational perspective in an implementation neutral fashion. The use of an OSI-SM or ODP-based technology is considered an engineering viewpoint decision that does not affect the system specification.

We could characterise the above approach as a "least common denominator" one, in which the OSI-SM framework is "pruned" to fit the ODP model. Despite its ODP orientation, [X703] recognises the fact that multiple object access through scoping and filtering and event dissemination through event filtering and EFDs may need to be exposed in the computational viewpoint. This leaves open the possibility for other potential mappings between OSI-SM and ODP. We will present such an approach in the rest of this chapter.

4.4.2 Mapping OSI-SM GDMO to CORBA IDL

The previous approach for describing OSI-SM in ODP terms assumes that it is possible to map a managed object specified as a GDMO information object to an ODP computational interface. In the absence of an ODP computational modelling language, we will assume that this language is the CORBA IDL and we will consider the issues of mapping GDMO to CORBA IDL. This is in accordance with the fact that OMG CORBA is considered as the pragmatic counterpart of ODP and the fact that OMG specifications may evolve into relevant ODP recommendations.

The issue of mappings between GDMO and CORBA IDL has been addressed by the Joint Inter-Domain Management (JIDM) task force between the NMF and X/Open. Though the main intention behind this work was to result in the specification of generic gateways between different management technologies, the same principles and mappings can be used to support pure CORBA-based management systems. The JIDM work started in 1993 and the first important outcome was the comparison of the Internet SNMP, OSI-SM and OMG CORBA object models described in [Rutt94]. The *specification translation* aspects followed [JIDM95], including the generic mapping of GDMO to CORBA IDL. Since then, the focus has been the dynamic *interaction translation*, which can be used to support the construction of generic application-level gateways. We discussed "indirectly" some of the aspects behind mapping GDMO to IDL in section 4.3.3.2, while discussing the CORBA information model. We discuss again and elaborate on those aspects below.

While IDL interfaces have attributes in a similar fashion to GDMO objects, it is not possible to map GDMO to IDL attributes directly. This is because IDL attributes have only *get* and *set* properties, while GDMO attributes have additional *setToDefault*, *add* and *remove* properties. In addition, it is not possible to define specific exceptions associated with access to attributes in IDL, while this is possible in GDMO. As such, GDMO attributes should map to access methods in accordance with the relevant properties e.g. <attr>_get, <attr>_set, <attr>_setToDefault, <attr>_add and <attr>_remove.

While this approach solves partly the mismatch problem between GDMO and IDL attributes, it creates other problems. For example, it is not possible to interrogate the CORBA interface repository about the attributes an interface has in order to access those attributes through the dynamic invocation interface; the latter is useful for generic applications such as MIB browsers. In addition, attributes cannot be exported to the trader in the usual way, through the attribute name and type. GDMO attributes also exhibit "MATCHES FOR" properties which are used for filtering. There is no equivalent in IDL which means that filtering cannot be easily supported. We will revisit the issue of filtering in ODP-based management environments later in this section. In summary, there are problems with respect to the mapping of GDMO attributes to equivalent IDL constructs, emanating from the superior expressiveness of GDMO. The only proper solution would be for OMG to reconsider the IDL attribute model and "upgrade" it to an equivalent of the GDMO one.

GDMO actions can be naturally mapped onto IDL methods with input argument the action information and output argument the action result. Action parameters, which signify action-specific errors, are mapped onto IDL exceptions. GDMO notifications can be mapped onto separate interfaces that should be supported by managing objects and event channels. Two separate interfaces should be generated for the notifications of a managed object class, one for the "push" and one for the "pull" model. Notifications should be typically mapped onto oneway operations, for non-confirmed notifications, and onto normal operations, for confirmed notifications. In general though, the key problem with the mapping of notifications is that they result in different interfaces, fragmenting the relevant specification. In summary, the CORBA support for notifications from a specification point of view is not satisfactory. In addition, the notification dissemination model is less powerful than the OSI-SM, as it was discussed in section 4.3.3.3.

An additional difference between GDMO and CORBA IDL concerns the dynamic binding of functionality to managed object instances through conditional packages. This is a key feature of GDMO, used very often by information model designers, while it is not supported in IDL. The only solution is to make the functionality of GDMO conditional packages "mandatory" from a specification point of view i.e. their attributes and actions will be part of the resulting IDL interface. Their presence though will become an implementation issue. CORBA supports a standard *not_implemented* exception which will be raised whenever a method of a non-supported package is invoked. An interface should "advertise" the supported conditional packages through the *packages* attribute of the *i_top* interface, which will result from the translation of the OSI-SM *top* class [X720].

This approach is all that can be done regarding the mapping of conditional packages and has a number of problems. First of all, the functionality of assigning packages dynamically to object instances at creation time is lost. A more important problem is that conditional packages are sometimes used by information model designers in a fashion that cannot be supported through "hardwired" implementations. For example, the monitorMetric class specified in [X739] uses conditional packages in such a way that their inclusion (or not) has an impact on the functionality of the relevant object instance. While this is arguably poor modelling practice, such cases have often been used by information model designers. The only solution to this problem is to use non generic mappings but then generic application gateways between OSI-SM and CORBA cannot be supported. Another solution would be for ISO/ITU-T to "correct" any existing specifications that present this problem and instruct information modelling groups on the proper use of GDMO conditional packages, in an IDL-compatible fashion.

These are the rules for mapping GDMO to CORBA IDL, as specified by the JIDM group [JIDM95]. The author added his own views and opinions regarding various aspects of this mapping. While the latter goes a long way towards reconciling the two models, some semantics are inevitably lost in the translation. The most important problems concern the mapping of GDMO attributes to CORBA attributes, the mapping of notifications onto separate "unrelated" interfaces and the problem of conditional packages. In addition, CORBA does not allow the redefinition of attributes and operations in derived interfaces, so true polymorphism cannot be supported. This presents another mapping problem since this useful feature is often used in GDMO specifications. In the absence of true polymorphism, allomorphism is also meaningless in CORBA. We will ignore these problems at present, assuming that future CORBA versions may address the problem of a fully object-oriented IDL that supports polymorphic operations.

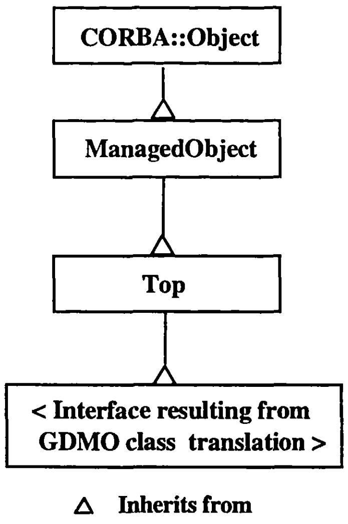


Figure 4-8 IDL Inheritance Hierarchy Resulting from GDMO Mapping

Having presented the rules for IDL to GDMO translation, it is possible to map OSI-SM GDMO managed objects to CORBA IDL interfaces and preserve all the work that has gone into the relevant OSI-SM / TMN specifications. The relevant translation may support gateways between CORBA and OSI-SM / TMN applications. It may be also used to support the native operation of management systems entirely in CORBA, as will be investigated in the rest of this chapter. The equivalent IDL interfaces follow exactly the same inheritance lattice as in GDMO, while the *i_top* interface is equivalent to the OSI-SM *top* class [X720]. The i_top interface inherits from the *i_ManagedObject* one, which in turn inherits from CORBA's Object, as do all the IDL interfaces. The resulting inheritance hierarchy is depicted in Figure 4-8.

The i_ManagedObject interface may support functionality common to all the managed objects, such as getting an object's name, accessing a number of attributes with one operation, evaluating a filter and returning the interface references of its superior or of its subordinate objects in the containment hierarchy. The latter is similar to a part of the functionality of the MO class in OSIMIS, as described in Chapter 3.

```
interface i_top : i_ManagedObject
{
    string            objectClass_get ();
    string            nameBinding_get ();
    StringList_t      packages_get ();          // conditional
    StringList_t      allomorphs_get ();        // .. (but not used)
};

interface i_uxObj : i_top
{
    SimpleNameType_t  uxObjId_get ();
    UTCTime_t         sysTime_get ();
    string            wiseSaying_get ();
    void              wiseSaying_set (in string);
    string            wiseSaying_setDefault ();
    ObservedValue_t   nUsers_get ();
    void              echo (in string, out string);
};
```

Code 4-1 The Top and UxObj Managed Object IDL Interfaces

The Code 4-1 caption above shows the potential mapping of the OSI-SM top class [X721] and the uxObj class used in the examples of Chapter 3, following the rules presented above which are largely based on the JIDM approach. The translation of the uxObj class will also comprise a notification interface and a factory interface which are not shown above. The GDMO to IDL translation assumes that ASN.1 data types for attributes, actions, notifications and parameters (i.e. exceptions) are also translated to CORBA IDL. [JIDM95] specifies the rules for ASN.1 to IDL translation. A complete specification of those classes in both GDMO and CORBA IDL can be found in Appendix C.

4.4.3 An Initial Mapping of the OSI-SM Model to CORBA

Having discussed the mapping of GDMO to CORBA IDL, we will consider how CORBA can be used instead of OSI-SM as the basis for TMN systems. We will consider first a pure ODP approach, which uses trading for object discovery and disregards completely hierarchical names based on containment relationships. This is the approach advocated often by ODP enthusiasts but there has never been an attempt in the literature to consider the issues involved. We will thus try to investigate if and how this approach can be used to support telecommunications management systems. We will then present another, more pragmatic approach, which uses name services and hierarchical containment relationships instead of trading. The second approach will be extended further to include TMN aspects such as object clustering, scoping, filtering and EFD-based event dissemination, migrating essentially the OSI-SM / TMN framework over CORBA.

4.4.3.1 A ODP-Oriented Approach Using Discovery Through Trading

We will assume first that GDMO specifications are translated to CORBA IDL as described above. If CORBA is used as the underlying access and distribution mechanism, managed objects will be realised as CORBA objects. The key difference is that clusters of managed objects logically bound together, e.g. objects representing various aspects of a managed network element, are not now seen collectively through an agent. As such, an important issue is to provide object discovery facilities which in TMN environments are supported first through directory access and then through scoping and filtering. Such facilities are very important in management environments where many instances of the same object type typically exist, with names not known in advance, e.g. call objects.

We will assume that containment relationships are treated just as any other relationship and that they are not used for naming. An alternative naming scheme might exist but this is not important since the solution is based on trading rather than naming. The trader will provide facilities similar to filtering based on assertions on object properties and attributes. It should be mentioned that trading services are still under development by OMG. An early implementation of the ODP trader specifications using CORBA has been used in the ACTS VITAL project, provided as background information to the project by Alcatel Alsthom Recherche [Trate96]. This particular implementation supports only equality checking on attributes and properties. This functionality is inferior to the OSI-SM filtering but we will assume that traders will eventually support more sophisticated assertions.

In a distributed management environment, managing objects will discover managed objects based on assertions on properties and attributes through the trader. An important issue with the use of trading is that federation is a key aspect in order to achieve scaleable systems. In essence, it will be necessary to have dedicated traders for every logical cluster of managed objects, for example in every managed element, in order to reduce traffic and increase real-time response. Those "low-level" servers will need to be unified by "higher-level" servers in a hierarchical fashion. Federation issues become thus very important but have not yet been worked out and are not simple. Even with such facilities in place, the management traffic in terms of the required application-level packets will be at least double that of OSI-SM. In CORBA, matching object references will be returned by the trader to the client object and the operations will be performed on an object-by-object basis. In OSI-SM, a multiple object access request will be sent in one packet while the results will be returned in linked replies, one for each object accessed. The use of CORBA for network management through federated trading is depicted in Figure 4-9.

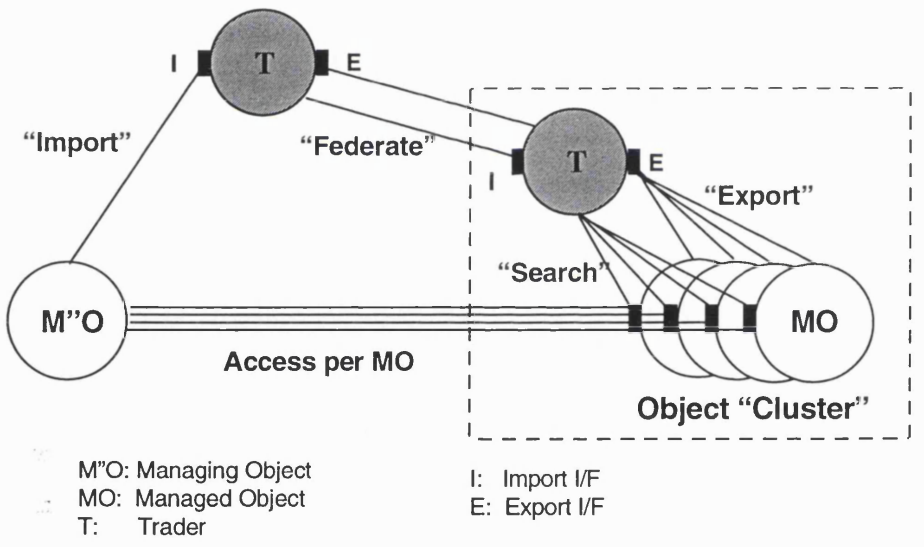


Figure 4-9 Object Discovery Through Trading

We will consider now the same example we considered in section 2.2.4 of Chapter 2 for OSI-SM in order to demonstrate in more detail the operation of this CORBA-based framework. The manager in this case will be a CORBA client object which will have to discover the routing table entries in a router that point to a particular "next hop address". through the trader. We will assume that there exists a trader in the router's node so that interactions between the router managed objects and that trader do not incur management traffic. This trader together with other traders will constitute a federated trader network, which has similarities to the global OSI Directory [X500]. The difference is that federation in the directory is based on hierarchical information structuring principles while the global trading "service space" does not typically exhibit hierarchical properties and, as such, federation is much more difficult to provide.

Objects in the router will have to export their properties and attributes to the trader when they are created and withdraw them when they are deleted. The interface reference of the local trader is supposed to be a "well known" parameter. The local trader should tell "superior" or "neighbour" traders about the service space it administers. This should be done for whole groups of objects with particular properties rather than on a per object basis for scalability reasons. In our example, the local trader might "announce" that it administers route table entries with the *routerId=router-A* property. This property does not map to a route attribute of the relevant interface but this is not necessary in CORBA: any properties may be dynamically associated to interfaces. These interactions are shown as "export" and "federate" in Figure 4-9.

When the manager object wants to access those route objects, it will have to contact its local trader and ask for interfaces with particular properties. The relevant operation could be expressed as: *import (ifType=routeEntry, properties={router=router-A, nextHopAddr=X}).* The trader should scan the relevant properties and based on the fact that *router=router-A* has been specified it should federate the request to the trader in the router. This federation may also involve other intermediate traders but only the two traders mentioned are shown in Figure 4-9. The trader in the router will return the relevant interface references to the initial trader which will subsequently return them to the manager. If dynamically changing attributes are involved in the assertion, the trader may have to "search" through the relevant objects and retrieve the values of those attributes; this is not the case in the example presented here.

Having retrieved the interface references, the manager will perform the operations to the relevant route entries according to its management policy e.g. retrieve the route destination and quality attributes, delete the routes, etc. These operations will have to be performed concurrently in order to minimise the overall latency. Given the fact that CORBA operations are performed synchronously, a multi-threaded execution environment is of paramount importance. The minimum management traffic incurred will be: 4 RPC packets for trading (2 to/from the domain trader and at least another 2 between the latter and the trader in the router); and 2*N RPC packets for retrieving the entry attributes i.e. 2*(N+2) packets in total. The equivalent amount of traffic using OSI-SM was N+2 packets as described in section 2.2.4 of Chapter 2. On the other hand, the amount of data in those CORBA packets may be less than in the equivalent OSI-SM packets. We will investigate the exact amount of traffic incurred by CORBA in section 4.5.3.

We will emphasise again the fact that this solution is only hypothetical at present, assuming traders that support rich query facilities, and more important, federation. As such, it can not be provided with the current state of ODP / CORBA technology. Supporting trader federation in an optimal fashion is an interesting research issue that needs to be addressed. The analysis in this section has been presented in [Pav97a] and [Pav97d].

4.4.3.2 A TMN-Oriented Approach Using Discovery Through Naming and Containment

While in the previous approach we adopted a pure ODP view, we will now look at another, TMN-oriented approach that can be thought as being at the other end of the spectrum.

This approach assumes the same hierarchical naming scheme as in OSI-SM / TMN systems, based on the GDMO prescribed name bindings in "agent" domains and on the OSI Directory global name schema specified in [X750]. For example, the name of the root object in a CORBA managed object cluster that constitutes a TMN OS could be:

{ c=GB, o=UCL, ou=CS, cn=NM-OS, systemId=NM-OS }

This is now an instance of the CORBA CosNaming::Name IDL type as specified by the OMG Name Service [COSS]. Both OMG and OSI-SM names are ordered lists of *type=value* components, so there can be a direct mapping between the two. The key difference is that the OMG name space is generally an acyclic graph instead of a hierarchical tree. Since we are adopting the TMN hierarchical naming principles, the OMG *management* name space becomes a hierarchical tree.

The first four components of the above example name denote *naming contexts,* i.e. they are not associated to CORBA objects. The fifth component, i.e. systemId=NM-OS, is a name bound to the compound context defined by the previous four. These contexts and the relevant name will be registered in the CORBA naming service [COSS]. A client or manager object will be able to resolve the object's name to an interface reference through the naming server. In addition, the client will be also able to discover all the management applications running in the UCL CS domain by performing a *list* operation on the naming context *{ c=GB, o=UCL, ou=CS }*. It will subsequently be able to obtain the subordinate object names of those contexts, e.g. *systemId=NM-OS* for the *cn=NM-OS* context, and resolve them to interface references. This architecture provides discovery functionality similar to that of the OSI Directory in OSI-SM / TMN environments [X750]. In addition, it can be supported by the current state of CORBA through the use of naming services [COSS]. The latter may be federated in order to cope with large name spaces and different administrative domains. Federated name services can be easily provided in comparison to federated trading because of the hierarchical structure of the name space.

Having presented the system discovery aspects, we also need to address discovery facilities within an MIT cluster. Every managed object is aware of its name, which will be passed to it at by its "factory" at creation time. In addition, every managed object is aware of its superior and subordinate objects. Those object references form now a "virtual" MIT, since the relevant managed objects may be physically distributed across different network nodes. The superior reference is passed to an object at creation time. The subordinate references are passed to an object by the subordinate objects themselves, which update their superior at creation and deletion and maintain the "referential integrity" of the MIT. The Code 4-2 caption shows a potential IDL specification of the i_ManagedObject interface that supports this functionality.

```
interface i_ManagedObject
{
    CosNaming::Name   getName ();
    CosNaming::NameComponent
                      getRelativeName ();
    i_ManagedObject   resolve (in CosNaming::Name name)
                            raises (NotFound);

    i_ManagedObject   getSuperior ();
    ManagedObjectList getSubordinates ();
    oneway void       getSubordinatesAsync
                         (in i_ManagerObject manager, in long invId);

    void              addSubordinate (in i_ManagedObject subord)
                            raises (InvalidObject);
    void              removeSubordinate (in i_ManagedObject subord)
                            raises (InvalidObject);

    void              destroy ()
                            raises (NotDeletable,
                                    DeleteContainedObjects);
};
```

Code 4-2 The i_ManagedObject IDL Interface

Every managed object provides access to the references of its superior and immediately subordinate objects in the MIT. It can *resolve* a subordinate name to a reference by using recursively the *getSubordinates* and *getRelativeName* methods.

Manager objects may discover the exact structure of the MIT, starting from the root object and using those discovery facilities. This approach is in fact similar to the OSI-SM / TMN one apart from scoping and filtering. It should be noted that every object acts as a name server for objects in its subtree. The key advantage of the approach is that managed objects other than the MIT root do not have to register with the name service; this results in fewer interactions across the network and more timely operation. If the name service was used instead, subordinate names would have to be retrieved from the name server through a "list" operation and subsequently mapped to interfaces references through a "resolve" operation. In this architecture, the CORBA name service is only used for getting access to the root MIT object.

The problem with this approach is that the advantages of the ODP trading or OSI-SM discovery through scoping and filtering are lost. In the example presented before, the client object would have to retrieve the whole routing table, identify the desired routing entries and subsequently perform operations on them. In this respect, the approach is more akin to the Internet SNMP rather than to OSI-SM.

We could have added scoping at least to the i_ManagedObject interface, since it can be easily supported by traversing the "contains" relationship through the getSubordinates method. The problem though is one of "culture": scoping is a CMIS-related facility while the i_ManagedObject interface is totally unrelated to CMIS as an access method. Adding scoping, filtering and the full OSI-SM access functionality over CORBA is the next step. We will address the relevant issues in section 4.4.4.

It should be finally noted that the *getSubordinatesAsync* method is an engineering optimisation. Object references are fairly big as it will be discussed in the performance analysis section. As a result, the size of the GIOP response packet will be very big for the getSubordinates method. The getSubordinatesAsync method instructs the managed object to start "pushing" the references one by one in the opposite direction. The symmetric method for receiving the result is supported by the *i_ManagerObject* interface.

4.4.3.3 Adding Object Lifecycle and Event Dissemination Aspects

In the previous two sections we concentrated on the discovery and access aspects, adopting first a pure ODP and then a minimal OSI-SM approach over CORBA. Two other important aspects of a management framework are object lifecycle, i.e. managed object creation and deletion, and event dissemination. We will address those here, taking a realistic approach which uses existing CORBA facilities. As such, the relevant design completes the second approach presented above.

In every "agent" domain, there will exist a *factory finder* object, bound to the domain naming context e.g. cn=NM-OS. A client will be able to obtain its name from the name server through a "list" operation and resolve it subsequently to an interface reference. A further optimisation can be achieved by agreeing in advance on the relative name of the factory finders e.g. *factoryFinderId=NULL*, since there will always exist a single instance in a domain. The factory finder will provide access to specific factories for a particular type of interface, e.g. i_uxObj, as advocated in the CORBA lifecycle services [COSS]. Specific factory interfaces will exist for every GDMO class that has *create* properties. A factory interface will take parameters according to the GDMO class specification, which may include the name of the object to create, its superior's name, a "reference" object and initial values for attributes. A factory interface bears similarities to the CMIS *m-create* primitive but initial attribute values can be strongly typed. An example factory interface for the i_uxObj is shown in the Code 4-3 caption.

```
interface i_uxObjFactory
{
    i_uxObj  create_with_name(in CosNaming::Name uxObjName,
                                 in string wiseSayingValue)
                          raises (invalidName, duplicateName);

    i_uxObj  create_with_automatic_name
                                 (in string wiseSayingValue,
                                  out CosNaming::Name uxObjName);
};
```

Code 4-3 The i_uxObjFactory IDL Interface

Managed object deletion is supported through the *destroy* method of the i_ManagedObject interface. Derived implementation classes will have to redefine the relevant method behaviour according to the GDMO name binding properties i.e. deny deletion, instruct the caller to delete contained objects first or delete the whole subtree. As it was mentioned in section 4.3.3.2, operations in which the object destroys itself are not well supported in current CORBA implementations: a standard exception is generated and the client object does not know if the server object cleaned up and died properly or it left a mess behind. This behaviour will be hopefully corrected in future implementations.

The final important point for a complete CORBA-based architecture is event dissemination. This can be based on the existing OMG *event services* [COSS]. In every "agent" domain, there will exist a *channel finder* object, in a similar fashion to the factory finder one. This will provide access to event channels that serve specific event types. Managed objects that generate notifications will register with the corresponding event channels as "event suppliers". Manager objects will get access first to the channel finder through the naming service and then to the particular event-specific channels, registering as "event consumers". Generated notifications will be sent to the corresponding channels and will be subsequently passed to the manager objects using the push or pull model. The fact that event channels correspond to particular event types can support strongly typed event dissemination. Event type specific push and pull interfaces will be produced for every GDMO notification and will be supported by the relevant channels.

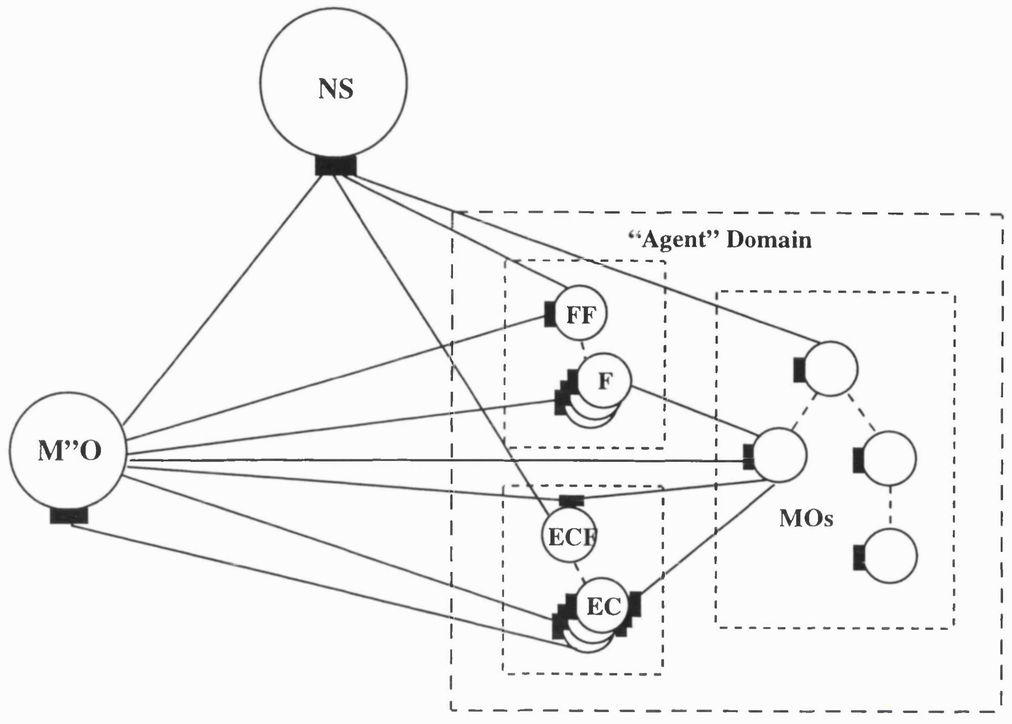
In summary, this event reporting approach adopts the CORBA event services. This means that the "agent" does not support EFD [X734] and log managed objects. The drawback is the functionality of event filtering and event logging are lost. Event logging could be provided by modifying event channels to get access to the MIT, check log objects and create log records through the relevant factory interface. In this case, the log filter should only contain assertions on the event type. While this approach is feasible, it requires the modification of the CORBA event channels [COSS].

Let's consider again the routing table example we presented before. The manager object is interested to know of any added or deleted routes that point to a particular next hop address. Assuming that *routeEntryCreation* and *routeEntryDeletion* events are defined, the manager will have to register with the relevant event channels. It will subsequently receive all the route entry creation and deletion events and will have to select locally those of interest, assuming that the next hop address property is contained in the notification. While this approach generates additional management traffic in comparison to the OSI-SM event model, it provides a workable solution based on the current OMG facilities.

4.4.3.4 Summary and the Proposed Architecture

In the previous sections we looked at the issues of using CORBA instead of OSI-SM as the base technology for TMN systems. We presented first a pure ODP-based approach for object discovery through trading. This cannot be supported with the current state of technology because OMG trading services are not finalised, they do not support rich query facilities and federation issues have not been addressed. We then presented a OSI-SM-based discovery approach, which uses hierarchical naming and containment relationships while the OMG naming service is used to locate the root object, the factory finder and the event channel finder in every "agent" domain. In this approach OMG event services support event dissemination. The relevant architecture is depicted in Figure 4-10, showing the various interactions as described in the previous sections.

The key advantage of using CORBA is that the managed objects that constitute a logical "agent" cluster may be distributed across different "capsules", i.e. operating system processes, which may in turn be distributed across different network nodes. The event channel finder and event channels will be located in the same capsule. The managed object factories will be located in the capsules where the relevant interfaces will be created. The factory finder will be statically configured to know the references of the relevant factories in that domain.



M"O: Managing Object
MO: Managed Object
NS: Name Server

FF: Factory Finder
F: Factory
ECF: Event Channel Finder
EC: Event Channel

Figure 4-10 A Basic Architecture for OSI-SM to CORBA Mapping

The approach presented is feasible with the current state of CORBA technology. The most difficult aspect to realise is the functionality of the i_ManagedObject interface. This is in fact a subset of the MO class of the OSIMIS GMS, which was presented in section 3.6 of Chapter 3. The disadvantages in comparison to OSI-SM are the following:

- there is no support for multiple attribute access;
- there is no support for multiple object access through one management operation;
- object discovery facilities do not support scoping and filtering;
- events are disseminated based on the event type i.e. there is no support for filtering;
- there is no support for logging; and
- non-DII operations on objects can only be synchronous, which requires support for multi-threaded operation.

In the next section we will examine how to address those disadvantages, reproducing the full OSI-SM functionality over CORBA.

4.4.4 A Complete Mapping of the OSI-SM Model to CORBA

Since the JIDM work on the mapping between GDMO and CORBA IDL was first published in 1994, a number of researchers started investigating the issues behind a CORBA-based management architecture. The work presented in the previous section is a part of the research work done by the author on this topic. The rest of the research work, which completes the proposed framework, will be presented in this section. We will examine first the related work by other researchers, presenting it in chronological order.

4.4.4.1 Related Research Work

[Fuen94] discusses the application of the TINA ODP-based architecture to management services. It presents the view that management applications should be modelled by OSI-SM-like agents, which are computational objects with IDL interfaces in the TINA management architecture. Managed objects do not have their own computational interface but are specified as information objects in Quasi-GDMO and realised as engineering objects within the agent.

The GDMO to CORBA IDL mapping presented in [JIDM95] addresses the static translation aspects. The architecture of a management environment based on the resulting CORBA specifications is another issue. [Mazum96] presents the first research work on such an architecture as a proposal to the JIDM group. The first version of this work appeared in 1995 and tries to re-use as much as possible the existing OMG services. The key element of this approach is that it establishes a "shared management knowledge" repository in CORBA, which recaptures aspects of the GDMO specification lost in the translation e.g. the MATCHES FOR properties of attributes. The OMG property service is used to support filtering in a complicated fashion while the TINA-specified notification service is used in conjunction with EFDs for event dissemination. The proposed approach is very complex and has not been adopted by the JIDM group.

[Geni96] is a PhD thesis titled "Towards a Distributed Architecture for Systems Management". It addresses the mapping of OSI-SM to ODP principles by using the JIDM translation approach, which it applies to ANSA instead of CORBA IDL. The presented concepts have been validated through the implementation of bidirectional gateways between ANSA and OSI-SM [Geni95]. The problem with the approach is that it takes a pure ANSA / ODP view, in which a lot of the OSI-SM functionality is lost, namely multiple attribute access, multiple operations to objects through scoping and filtering and fine-grain control of notifications through filtering and event logging.

The author started working in this area in 1995 as a result of his involvement in the ACTS VITAL project. The latter tries to validate the emerging TINA framework for service control and network management [TINA]. The author's initial approach was to model OSI-SM agents as computational entities with CMIS-like IDL interfaces, based on the initial ideas in [Fuen94] but taking those to completion as presented here. The first version of the relevant architecture and specification was released within the VITAL project in April 1996 [Pav96e]. As a result, the author was pointed to a TINA-C group working in a very similar direction and proposing a similar architecture [Garc96].

[Garc96] realises that the full functionality of OSI-SM needs to be preserved over distributed object frameworks because of its importance for telecommunications management environments. It proposes that managed objects are grouped together in "agent" clusters and named using TMN-based hierarchical naming principles. In addition, it proposes those to be administered by a Management Broker (MB), which is a computational entity similar to an OSI-SM agent. The latter offers a CMIS-like interface which supports multiple attribute access and multiple object access through scoping and filtering. Event reporting and logging are supported through EFD and log managed objects. This approach was only a paper exercise that never went into considerations behind the potential realisation of such a framework. As such, it was never taken any further within TINA-C.

The author architected a very similar approach which could be realised based on the OSIMIS environment and its reusable software components. A first implementation of a generic gateway between CORBA and OSI-SM was produced in the summer of 1996 by T. Tin of UCL and was used in the VITAL project [Pav97b][Huel97]. At the same time, the author was discussing this approach with the TMN research group of the Hewlett-Packard Research Laboratory at Bristol (HPLB). One of the HPLB researchers supervised a MSc project at the University of Twente, which verified the feasibility of the framework with another implementation [Hars96].

Finally, [Schad96] proposes a model for distributed applications management which tries to extend CORBA with multiple object access and fine-grain event filtering facilities. Multiple operations are supported by "multicast objects" while the fine grain event model is based on a combination of EFDs with the CORBA event services.

Having presented all the relevant research work in this area, the obvious question is which approach did the JIDM group finally adopt for the CORBA to OSI-SM gateways. The answer is none of the above. [Hier96] is a relatively new proposal which has been adopted. While different from the other official JIDM proposal [Mazum96], it combines elements of the other approaches.

Managed objects are organised in "agent" domains and are named hierarchically. Event dissemination is handled through a specialisation of the OMG event service, using event channels in both manager and agent domains but with no filtering and logging at present. Multiple attribute and multiple object access is supported through the JIDM i_ManagedObject interface which is CMIS-like. This means that every managed object acts as an agent or management broker for its subtree. This approach is different to the [Pav97b] and [Garc96], we will discuss the benefit of a separate management broker later.

4.4.4.2 Adding Multiple Attribute Access and Filtering

We will now examine if and how the elusive full OSI-SM functionality can be added to the framework presented in section 4.4.3.4.

We will start first with multiple attribute access. With the current GDMO to IDL mapping, every attribute is mapped to one or more access methods. As a consequence, manager objects have to access attributes on a one-by-one basis, which creates unnecessary management traffic. Accessing multiple attributes is an important management requirement. In addition, many applications use the CMIS "get all attributes" facility, which should also be supported. The obvious place to put this functionality is the i_ManagedObject interface, but how can this be supported?

The key problem is knowing what the attributes of a managed object instance are. The i_ManagedObject part of a MO instance could interrogate the CORBA interface repository for the attributes of every derived interface and access them locally, through the DII. Unfortunately, this approach will not work. The problem is that as a result of the GDMO to IDL translation, the notion of attributes is lost. This means that the CORBA interface repository cannot be used. An alternative approach would be to provide "shared management knowledge" about the information of a GDMO-derived IDL interface. For example, this information is stored in a *discovery* managed object in OSI-SM / TMN environments [X750]; [Mazum96] proposes such an approach.

A third and most efficient approach would be similar to that of OSIMIS and most TMN platforms: every derived implementation class should pass the names of its attributes to the i_ManagedObject part of an instance at creation time. The only problem with this approach is that this code will need to be hand-written, which is both tedious and error prone. In TMN platforms, this code is automatically produced by the GDMO compiler. A way around this problem would be the existence of special "JIDM-aware" IDL compilers which could produce this code automatically.

In summary, it is possible to support multiple attribute access. The example method signatures for getting multiple attributes are shown in the Code 4-4 caption. A similar *setAttributes* method could also be provided. It should be finally noted that the resulting methods are weakly-typed because the IDL *any* type is used for attribute values.

```
typedef string AttributeId_t;
typedef sequence<AttributeId_t> AttributeIdList_t;

struct Attribute {
    AttributeId_t attrId;
    any           attrVal;
};

enum GetListError_t {
    noError,
    noSuchAttribute
};

struct GetAttribute_t {
    GetListError_t error;
    AttributeId_t  attrId;
    any            attrVal;      // "empty" in errors
};
typedef sequence<GetAttribute_t> GetAttributeList_t;


interface i_ManagedObject
{
    // . . .

    void     getAttributes     (in  AttributeIdList_t  attrIdList,
                                out GetAttributeList_t attrList);

    void     getAllAttributes (out AttributeList_t attrList);


    boolean  evaluateFilter (in Filter_t filter);

};
```

Code 4-4 Multiple Attribute Access and Filtering

The next aspect to consider is filtering, which is a much more difficult proposition. [Mazum96] proposes to use the OMG *property* service, together with "shared management knowledge" which provides access to the GDMO MATCHES FOR properties of attributes. The solution is very complex, not clearly presented and has not been validated through implementation. [Hier96] specifies filtering as part of the CMIS-like access methods of the i_ManagedObject interface but does not discuss at all how it is going to be provided. It should be noted that supporting filtering in CORBA to OSI-SM gateways is easy since the filter will be evaluated in the target OSI-SM agent; this is not the case in native CORBA environments.

Let's revisit first how filtering is supported in OSI-SM environments and examine how the same functionality could be provided in CORBA. Filter assertions on a particular attribute are evaluated by the attribute itself. The ASN.1 compiler produces a *compare* method which tests the data type for equality, while ordering and substring testing have to be added by hand. In multi-valued attributes, equality concerns the contained elements. Produced classes derive from a generic class, i.e. *Attr* in OSIMIS, which can evaluate a filter assertion by using the compare method and methods to "walk through" a multi-valued attribute. This was described in section 3.4 of Chapter 3. The generic MO class can evaluate filter expressions since it keeps "handles" to attributes of derived classes, which evaluate specific assertions in the filter.

The problem with CORBA is that attributes are not "first class citizens" of the framework. Defining an attribute in an IDL interface results in nothing more than access methods being produced, without any special support for the relevant data type. As such, there is no automatic support for equality comparison and subsequently for the evaluation of filter assertions. One solution to this problem would be for OMG to consider providing such support through a special extension to IDL. Types preceded by some special keyword, e.g. `attribute`, could be treated specially, deriving from a generic attribute class and supporting equality assertions. This requires though the modification of both the IDL and the relevant programming language mappings.

[Mazum96] mentions that the comparison methods required for filtering could be either provided by hand, which is obviously not desirable, or produced by modified IDL compilers which understand comment lines with special significance. The Alcatel trader implementation [Trate96] uses string property values in order to be able to support equality testing. It also supports ordering assertions on attributes but only for string, integer and real types. These are "cast back" to the precise type according to the *type code* that characterises instances of the CORBA *any* type.

In summary, it is not easy to provide filtering in native CORBA environments. In general, the mapping of IDL types to object classes is not rich enough, lacking support for comparison, pretty-printing and other generic functionality. As such, it is problematic to deal with instances of the *any* type. This is an area that needs special attention by the OMG if CORBA is to become the ubiquitous object infrastructure. We will assume for the time being that filtering is possible and that the i_ManagedObject interface supports the *evaluateFilter* method as it was shown in the Code 4-4 caption This is in accordance to the OSI-SM management information model [X720] which suggests that filtering should be supported by a managed object. The Filter_t is the IDL type that maps to the CMISFilter ASN.1 type of [X711] according to the JIDM rules.

Given the support for filtering and the fact that containment relationships can be navigated through the *getSuperior* and *getSubordinates* methods, multiple access to managed objects and sophisticated discovery facilities can be provided. The relevant functionality is similar to that provided by OSI-SM agents and the question is where it should be placed. Both [Mazum96] and [Hier96], i.e. the two JIDM proposals, suggest that it should be part of the i_ManagedObject interface. This essentially means that every managed object behaves as an agent for its subtree. In contrast, the author [Pav97b][Pav97d] and [Garc96] propose to separate this functionality from the managed objects, so that different sophisticated access styles can be provided. A detailed analysis of the second approach is presented in the next section.

4.4.4.3 Fine-grain Event Dissemination and Multiple Object Access Through the Management Broker

The i_ManagedObject interface supports the following functionality as presented in the previous sections:

- name resolution for contained objects;
- access to the "containedIn" and "contains" relationships;
- filter evaluation;
- get and set access to multiple attributes through a single access method; and
- object deletion.

This functionality is exactly as prescribed by the OSI-SM MIM [X720] and supported in the OSIMIS by the GMS MO class. This approach essentially "externalises" a managed object which now becomes "first class citizen" of the proposed distributed management framework.

ODP purists may argue that some of the above managed object functionality is "OSI-SM oriented" and not in the spirit of ODP, in particular the hierarchical naming, containment relationships and filtering. Hierarchical naming schemes are used in many contexts and not only in OSI-SM. For example, the postal and network addresses and even the OMG naming architecture use hierarchical schemes. Managed objects may exhibit many other relationships in addition to containment. These can be realised either through "pointer" attributes, in a similar fashion to containment, or through separate relationship objects [X725] that map onto the OMG relationship service [COSS]. Finally, filtering provides a powerful mechanism to express sophisticated assertions on attributes and properties. The ODP / OMG trading service might also support filtering in the future.

Given the support for filtering, fine-grain event reporting and logging can be provided by migrating the relevant OSI-SM models over CORBA. In every "agent" domain, there will exist a Event Processing (EP) object. Managed objects will get access it through local means, e.g. the factories may pass its reference to MOs at creation time. MOs will subsequently "push" their notifications to it. The EP object will create the "potential event report / log record" through the relevant object factory, evaluate the filters of EFDs and logs and may instruct the latter to send the event or log it as a log record accordingly. This is exactly the behaviour prescribed in [X734] and [X735]. Note that the existence of the EP object is totally transparent to manager objects that are interested to receive event reports.

Manager objects will request the forwarding of events by creating EFDs and setting the *destination* attribute to contain either their name or object reference. This implies that the syntax of the *destination* and *backupDestinationList* EFD attributes [X734] will have to be slightly modified. Destinations are currently specified as OSI-SM distinguished names which are equivalent to CORBA names, but CORBA object references should be added. This approach retains the full power and expressiveness of the OSI-SM event reporting and logging. A pictorial view of this approach is presented in Figure 4-11.

The last aspect of the OSI-SM / TMN framework that needs to be provided is support for multiple object discovery and access facilities based on scoping and filtering. Such access facilities are certainly "OSI-SM / TMN specific" and should be provided in an incremental fashion, without being an integral aspect of the rest of the framework. A key reason for considering those separately is they are do not represent the only way of providing this type of functionality. For example, in the CMIS/P access model containment relationships are navigated first through scoping with filtering applied at the end of the selection process. [Pav97e] proposes an enhanced model in which any relationships could be navigated, with filtering possibly applied at various stages of the object selection process. Other mechanisms may be invented in the future that suit best the needs of particular management environments.

This is the why the author proposes to separate the CMIS-based access aspects from the rest of the management framework. As such, CMIS-based access should *not* be part of the i_ManagedObject interface but should be supported by a separate *Management Broker* (MB) object. Given the fact that CMIS is the current access mechanism in TMN environments, a MB should always exist in an "agent" domain with its name bound to the domain naming context e.g. *{ c=GB, o=UCL, ou=CS, cn=NM-OS, brokerId=CMIS }*. Managed objects could be accessed either directly or through the MB. The advantage of MB-based access is object discovery and multiple object access through scoping and filtering. The disadvantage is that the relevant access paradigm is weakly-typed: attribute and action values are of the IDL *any* type. The architecture of the proposed framework is depicted in Figure 4-11, including the event dissemination through EFDs. This updates the architecture that was presented in Figure 4-10.

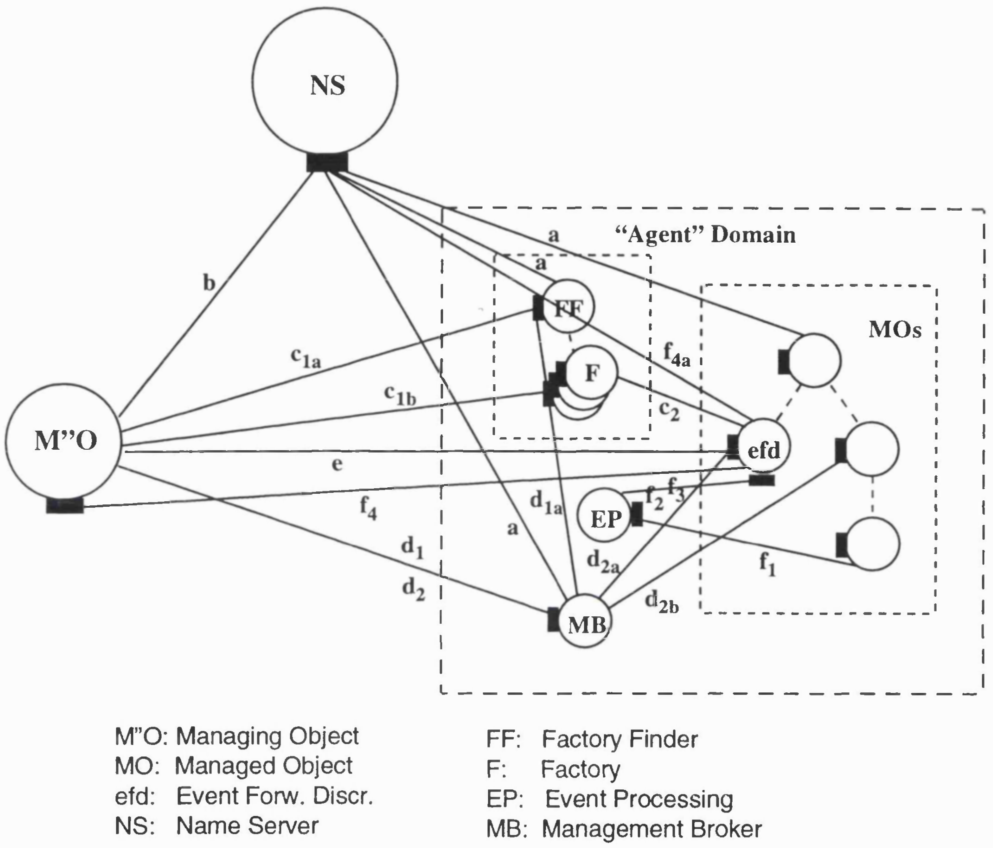


Figure 4-11 A Complete Architecture for OSI-SM to CORBA Mapping

When an "agent" domain is instantiated, the root MIT MO, the factory finder and the management broker register themselves with the name service (interactions *a* in the figure). Manager objects need to know in advance the domain name, e.g. *{ c=GB, o=UCL, ou=CS, cn=NM-OS }*. They may invoke a list operation on the name service and discover the names and subsequently the references of the MIT root, FF and MB objects (interaction *b*). A MO may be created either in a strongly-typed fashion through the relevant factory (interactions c_{1a} and c_{1b}) or in a generic, weakly-typed fashion through the MB (interactions d_1 and d_{1a}). The manager may subsequently access the object either directly (interaction e) or through the MB. In the latter case it will probably access more than one MOs e.g. to suspend the operation of all its EFDs (interactions d_2, d_{2a} and d_{2b}). A MO emits a notification by "pushing" it to the event processing object (interaction f_1). The latter will create first a "potential event report", retrieve an EFD's filter (interaction f_2) and evaluate it. The potential event report is not shown since it is manipulated locally by the EP i.e. can be thought as encapsulated by it. If the filter evaluates to true, it will instruct the EFD to send the event report (interaction f_3). The EFD may need to resolve the name of the manager to an interface reference through the name service (interaction f_{4a}) and "push" the event to the manager (interaction f_4).

In order to make the use of the framework more concrete, we will consider the same example we considered in section 4.4.3.1 and also in section 2.2.4 of Chapter 2. The manager object in this case will know the naming context of the router e.g. *{ c=GB, o=UCL, ou=CS, cn=router-A }*. By concatenating the relative name *brokerId=CMIS*, it will be able to obtain a reference to the broker object. It could then perform a *multipleObjectGetAsync* operation (shown later in the Code 4-6 caption) to the object with name *{ subsystemId=nw, protEntityId=clnp, tableId=route }* and request the first level subordinates that satisfy the filter *(nextHopAddr=X)*. The amount of traffic incurred in terms of packets will be exactly the same as in OSI-SM.

An alternative approach would be to perform the same operations without the management broker. The relative name of the root MIT object will be *managedElement=router-A*, so its name can be resolved to an interface reference through the name server. A subsequent resolve operation to the managedElement object will return an interface reference to the table object and then a *getSubordinates* operation will retrieve the subordinate references. The route entries will then be accessed one by one in order to identify the ones with the particular destination address. The problem in this case is that the traffic incurred will be much more than the previous approach, mainly because of the lack of filtering. In addition, a lot of object references need to be communicated and these are big as the measurements show in section 4.5.3.

Finally, the manager may request event reporting by creating an EFD object with the same filter as in OSI-SM. The EFD destination attribute may be initialised with the object reference of the manager and not only with its name. The advantage of the object reference is that it avoids name resolution through the name server, which is otherwise necessary. The EFD may be created in a weakly-typed fashion through the management broker, or in a strongly-typed fashion through the relevant factory. The reference of the factory will be obtained through the factory finder.

Returning to the proposed architecture, different management brokers may be specified and implemented in the future, providing varying styles of object discovery and multiple object access. With the advent of highly portable languages like Java [Sun96] and associated mobile code paradigms, managers may be able to install dynamically new management brokers in an agent domain to suit their needs. OMG is currently attempting to specify a framework for mobile code known as the Mobile Agent System Interoperability Facility (MASIF). The current management broker provides CMIS-based access since CMIS/P is currently the basis for the TMN Q_3 and X interfaces.

We describe below the broker's CMIS-like interface, examining the issues of mapping CMIS/P to CORBA IDL. The simplest form of management operations the MB provides are the CMIS m-get, m-set, m-action, m-delete and m-create, applied to a single managed object. These operations are also supported by the managed objects through the specific IDL interface that results from the GDMO to IDL translation, e.g. i_uxObj and i_uxObjFactory, and also through the generic i_ManagedObject interface. The reasons for providing the same functionality through the CMIS management broker are twofold:

a) we can exploit the CORBA *oneway* operations and provide an asynchronous interface - the interfaces resulting from GDMO to IDL translation according to the JIDM rules are only synchronous; and

b) the management broker may play the role of a CORBA-based management agent with the managed objects being plain engineering objects i.e. *without* IDL interfaces; we will examine this style of organisation in the next section.

The Code 4-5 caption shows the specification of the equivalent CMIS m-get and m-action methods in IDL, *get*, *getAsync* and *action*, *actionAsync* respectively. In the case of the asynchronous operation, the invoking manager object passes its reference for the result or error to be sent back. It also passes an invokeId argument, which can be used to distinguish between results / errors in the case of many outstanding invocations; this is equivalent to the CMISE/ROSE invokeId. The *objectClass* parameter may be used to force allomorphic behaviour. This is possible though only in environments like b) above, since allomorphism is meaningless for MOs with native CORBA IDL interfaces.

```
interface i_CMISBroker
{
    CosNaming::Name getName ();  // the broker's name

    // . . .

    void      get (
              in  CosNaming::Name     objectName,
              in  string              objectClass,    // for allomorphism
              in  AttributeIdList_t   attrIdList,     // empty -> "all"
              out GetAttributeList_t  attrList
              )
              raises (GET_ERRORS);

    onway void getAsync (
              in  i_CMISManager       managerRef,
              in  long                invokeId,
              in  CosNaming::Name     objectName,
              in  string              objectClass,    // for allomorphism
              in  AttributeIdList_t   attrIdList,     // empty -> "all"
              );

    void      action (
              in  CosNaming::Name     objectName,
              in  string              objectClass,    // for allomorphism
              in  string              actionType,
              in  any                 actionInfo,
              out any                 actionReply
              )
              raises (ACTION_ERRORS);

    onway void actionAsync (
              in  i_CMISManager       managerRef,
              in  long                invokeId,
              in  CosNaming::Name     objectName,
              in  string              objectClass,    // for allomorphism
              in  string              actionType,
              in  any                 actionInfo
              );
};
```

Code 4-5 Single Object CMIS-like Access in IDL

The next type of management operations are object discovery based on scoping and filtering and multiple object access for the m-get, m-set, m-action and m-delete operations. The synchronous version of the object discovery and the synchronous and asynchronous versions of the multiple object get and action operations are shown in the Code 4-6 caption. The *objectSelection* operation returns both the managed object's name and interface reference. In the case of environments like b) described above, only the name only is returned while the reference will be always "nil".

```
// Scope_t, Filter_t and Sync_t map exactly to the
// X.711 Scope, CMISFilter and CMISSync ASN.1 types

typedef struct ObjectSelection_t {
    Scope_t  scope;
    Filter_t filter;
};

typedef struct ObjectNameList_t {
    CosNaming::Name  name;
    Object           objectRef;
};

interface i_CMISBroker {

// . . .

    void     objectSelection (
                   in  CosNaming::Name    baseObjectName,
                   in  ObjectSelection_t objectSelection,
                   out ObjectNameList_t  objectNameList
             ) raises (OBJECT_SELECTION_ERRORS);

    void     multipleObjectGet (
                   in  CosNaming::Name    baseObjectName,
                   in  ObjectSelection_t objectSelection,
                   in  Sync_t            sync,
                   in  AttributeIdList_t attrIdList, // empty -> "all"
                   out GetResultList_t   resultList
             ) raises (MULTIPLE_OBJ_OPER_ERRORS);

    void     multipleObjectGetAsync (
                   in  i_CMISManager     managerRef,
                   in  long              invokeId,
                   in  CosNaming::Name   baseObjectName,
                   in  ObjectSelection_t objectSelection,
                   in  Sync_t            sync,
                   in  AttributeIdList_t attrIdList, // empty -> "all"
             );

    void     multipleObjectAction (
                   in  CosNaming::Name   baseObjectName,
                   in  ObjectSelection_t objectSelection,
                   in  Sync_t            sync,
                   in  string            actionType,
                   in  any               actionInfo,
                   out ActionResultList_t resultList
             ) raises (MULTIPLE_OBJ_OPER_ERRORS);

    void     multipleObjectActionAsync (
                   in  i_CMISManager     managerRef,
                   in  long              invokeId,
                   in  CosNaming::Name   baseObjectName,
                   in  ObjectSelection_t objectSelection,
                   in  Sync_t            sync,
                   in  string            actionType,
                   in  any               actionInfo,
             );

};
```

Code 4-6 Multiple Object CMIS-like Access in IDL

Finally, the Code 4-7 caption shows a part of the interface that should be supported by CMIS manager objects in order to receive asynchronous results and event reports. The latter may be both confirmed and non-confirmed. Note also that the event reports contain the name of the EFD that triggered them, so that manager applications may "demultiplex" them; this reflects an extension to CMIS/P that was proposed in Chapter 3.

```
interface i_CMISManager {

// . . .

    oneway void endOfLinkedReplies (
                in long invokeId
            );

    oneway void getResult (
                in long invokeId,
                in boolean linked,
                in GetResult_t result
            );

    oneway void actionResult (
                in long invokeId,
                in boolean linked,
                in ActionResult_t result
            );

    oneway void eventReportUnconfirmed (
                in  string           objectClass,
                in  CosNaming::Name  objectName,
                in  CosNaming::Name  efdName,
                in  UTCTime_t        eventTime,
                in  string           eventType,
                in  any              eventInfo    // optional
            );

    void    eventReport (
                in  string           objectClass,
                in  CosNaming::Name  objectName,
                in  CosNaming::Name  efdName,
                in  UTCTime_t        eventTime,
                in  string           eventType,
                in  any              eventInfo,   // optional
                out any              eventReply   // optional
            ) raises (EVENT_REPORT_ERRORS);
};
```

Code 4-7 The Generic CMIS-like Manager Interface

In summary, mapping CMIS to CORBA IDL interfaces is relatively straightforward. The author thought of the approach and produced the relevant specification in early 1996 [Pav96e]. [Garc96] and [Hier96] have also proposed similar conceptual approaches which mainly differ in the IDL mappings for CMIS. The author's approach tries to stay as close to CMIS as possible but introduces a number of simplifications, based on the OSIMIS design and implementation experience.

4.4.4.4 Design, Implementation and OSI-SM to CORBA Migration Aspects

In the previous sections we discussed the issues behind the mapping of the OSI-SM / TMN model onto emerging distributed object frameworks, using OMG CORBA as the target framework. We presented a complete architecture which is in the spirit of ODP / CORBA but which retains the power and expressiveness of OSI-SM, necessary for TMN environments. The proposed architecture uses only the OMG naming service from the common object services [COSS] and is implementable with the current state of CORBA, apart from automated support for filtering. The advantage of the CORBA-based architecture is that it will support the distribution of parts of TMN applications, which currently operate on a single network node. It might also be more performant that Q_3 protocol stacks; we will examine this assertion in the next section. Finally, a single technology will be used in TMN environments instead of the combination of OSI-SM [X700] and the OSI Directory [X500].

Given the target CORBA-based framework that was depicted in Figure 4-11, we will examine how this can be implemented. We will consider in particular a phased approach which reuses initially parts of the OSI-SM based infrastructure described in Chapter 3. Since OSI-SM is currently the base TMN technology, it is important to devise a phased transition strategy that will ease compatibility and interoperability with existing TMN systems and will reuse as much as possible existing TMN platform components.

The first step for migrating towards the target framework is to support only agent discovery and CMIS interactions through CORBA, without individual IDL interfaces for managed objects. This essentially means that the management broker will act as an agent that provides access to managed objects which are implemented by existing TMN platform infrastructure i.e. GDMO/ASN.1 compilers and relevant APIs. The MB may be used in conjunction to the existing Q_3 agent object within an agent application e.g. the CMISAgent object in the OSIMIS GMS. In this case, the TMN application in agent role will have two interfaces: the existing Q_3 interface and the CORBA version of the "Q_3" interface as specified by the i_CMISBroker and i_CMISManager IDL interfaces.

This minimal approach is depicted in Figure 4-13 and has no impact at all at the implementation of managed objects which are based on TMN platform technology e.g. OSIMIS. Existing OSI-based manager applications will continue to function while new CORBA-based management applications may be developed. CORBA manager objects get access to the MB interface reference through the CORBA naming services [COSS] while OSI manager objects get access to the presentation address of the OSI agent through the OSI directory [X750]. It should be noted that two different notations have been used in this figure to depict interactions, one for CORBA using object interfaces and one for OSI-SM using arrows. In addition, the CORBA interface notation belongs to computational viewpoint while the TMN agent application is depicted from an engineering viewpoint. This "mismatch" is necessary in order to avoid a more complicated drawing.

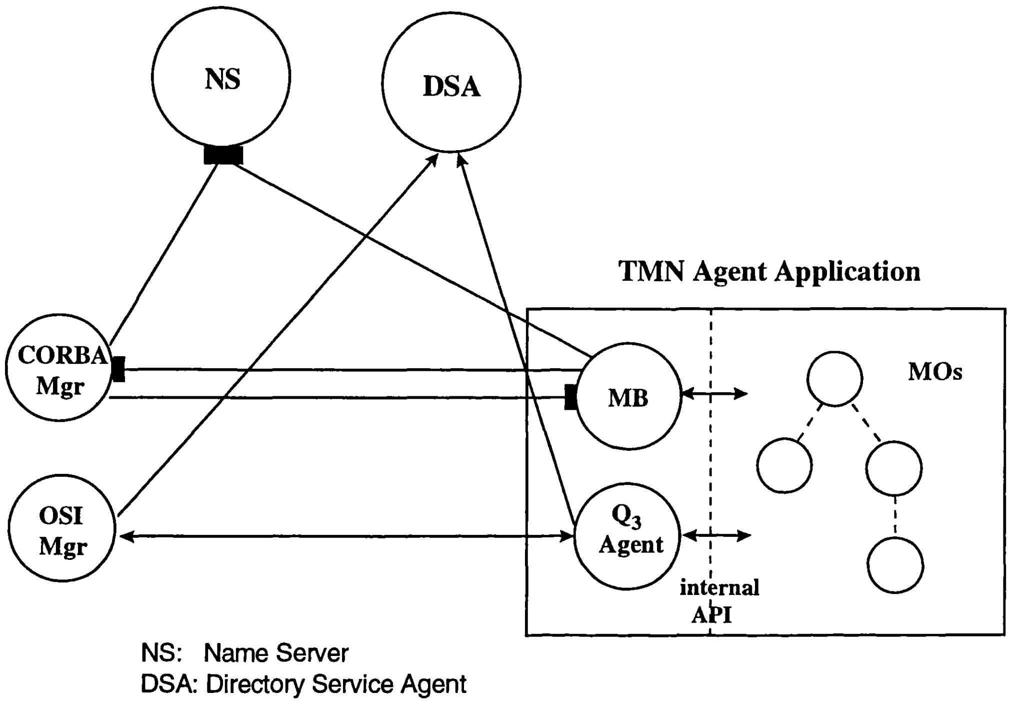


Figure 4-12 Dual Q_3 and CORBA Agent

This architecture exploits the fact that the object models are the same in the two frameworks, and provides a "dual-agent" access paradigm. Dual agents support different access methods for semantically equivalent managed objects [Newk94]. In this framework, the managed object GDMO/ASN.1 specifications are translated to IDL but the resulting IDL interface specifications are not instantiated: they simply "document" the management broker interface which provides access to those objects in a dynamic, weakly-typed fashion. The attribute, action and event ASN.1 types are used across the management broker interface through the equivalent IDL types.

A variation of this approach is the gateway approach, in which the management broker becomes a separate application which accesses one or more management agents in the "back-end" through their Q_3 interfaces. The gateway approach is useful to provide adaptation for TMN applications that are already deployed, in which case it is not possible to add to them the management broker in a tightly-coupled fashion. The disadvantage of that approach is reduced performance, which is in general the case with adaptation mechanisms.

T. Tin of UCL together with the author implemented a gateway version of the MB during the summer of 1996. This was subsequently used in the first trial of the VITAL project in October 1996. This was the first CORBA to CMIS/P gateway that provided the full OSI-SM functionality. Since then, TMN platform vendors have announced similar products based on the emerging JIDM proposal for the interaction translation [Hier96]. The author subsequently designed and implemented the tightly-coupled dual agent approach depicted in Figure 4-12. [Hauw97] also reports a very similar dual-agent approach. CORBA-based agent applications were used in the second trial of the VITAL project [Pav97b] and the first trial of the REFORM project [Sartz97].

Implementing the gateway was fairly straightforward. The only difficulty encountered was the bi-directional translation between ASN.1 and IDL data types. This can be automated if one has access to a flexible ASN.1 compiler, which could be customised to produce the equivalent IDL types as well as the conversion routines in both directions. Since OSIMIS uses the ISODE pepsy compiler which cannot be easily customised, these conversions had to be hand-coded. Implementing the tightly-coupled dual agent version was also straightforward given the well-defined OSIMIS APIs for accessing managed objects within an agent application. Based on this approach, existing OSIMIS agent applications can be made to work over CORBA by changing a few lines of code in the main program and re-linking them with the management broker library.

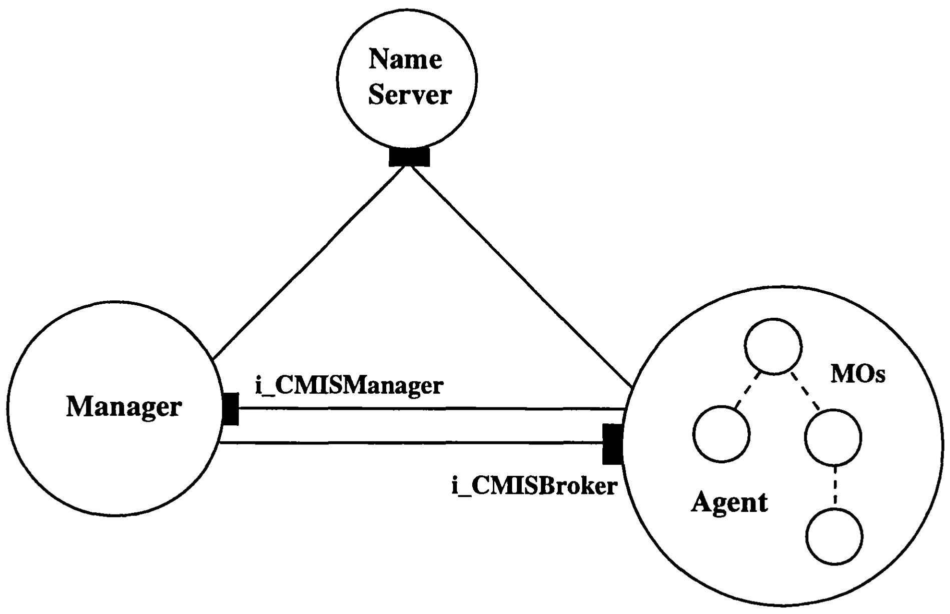


Figure 4-13 The Management Broker as an Agent Computational Object

Another way to look at this paradigm in which managed objects are accessed indirectly through a management broker is depicted in Figure 4-13. In this case the managed objects can be thought as encapsulated by the MB. The latter is a computational object in ODP terms which provides the agent aspects of the CMIS service through the i_CMISBroker interface. The manager supports the manager aspects through the i_CMISManager interface. Managed objects are specified in GDMO in the information viewpoint but do not manifest themselves in the computational viewpoint. They become directly engineering objects administered by the MB, in a similar fashion to OSI-SM agents. This is another view for the mapping the OSI-SM model to ODP.

The complete CORBA-based framework requires also that individual managed objects become computational constructs with IDL interfaces. The advantage in this case is strongly-typed access, superior distribution aspects and better performance as will be presented in section 4.5.2. Having produced the relevant specifications using GDMO to IDL translation, the key issue is how to support the functionality of the i_ManagedObject interface. The difficult aspects regarding this realisation are the "get all" attributes and the attribute filtering; we explain how these can be implemented below.

Classes for derived interfaces should pass to their i_ManagedObject parent class the names and types of their attributes when an object instance is created. The i_ManagedObject part will subsequently access the attributes of derived parts as if they belonged to separate objects, through the dynamic invocation interface. This scheme supports the multiple attribute get and set methods.

The only way to deal with filtering is to provide the IDL compare and "multi-valued" attribute traversal routines by hand, in a hash table indexed by the relevant type. The i_ManagedObject part will retrieve the attributes involved in the filter and will invoke the relevant compare and traverse methods, getting access to them through the attribute type which is known from its "repository". While this is feasible, it is far from desirable since it requires hand-written routines for every attribute type. On the other hand this is the only approach to support filtering until OMG considers the inclusion of relevant features in IDL and their concrete support through the programming language mappings.

Despite the fact that the author has not validated the above concepts through implementation, they are largely based on the design concepts of the managed objects in the OSIMIS GMS, which *have* been validated through implementation.

4.5 Performance Analysis and Evaluation

In this section we attempt a performance analysis and evaluation of the CORBA-based framework. We measure performance aspects of both native CORBA managed objects and of the management broker introduced in the previous section. Since one of the main motivations for migrating to CORBA is a potential performance advantage over OSI-SM, we attempt to verify if this is the case. Another potential advantage is the superior distribution framework and we attempt to evaluate its cost in terms of computing resources.

The measurements address the same aspects as for the OSI-SM framework, namely program size, response times and amount of communicated information. A commercial CORBA platform was used for the development and measurements. As such, it was not possible to measure aspects "beneath" the APIs available to the developer. This means is was impossible to scrutinise the end-to-end performance and attribute it to various parts of the infrastructure as we did for OSI-SM.

The measurements of the CORBA framework also aim to put into perspective the equivalent OSI-SM measurements. While we assessed the OSI-SM framework as performant given the low response times and the relatively modest memory requirements, those figures were absolute and were judged in relatively subjective terms. In this section, we compare and contrast those figures to the equivalent CORBA ones, so we will be able to re-evaluate the performance of OSI-SM in the context of this comparison.

The environment and methodology for the experiments were similar to those used for the OSI-SM experiments. These have been described in section 3.8.1 of Chapter 3. The *Orbix 2.2* implementation of CORBA was used, a leading CORBA product marketed by IONA technologies. Since there exist various programming language mappings for CORBA, the C++ mapping was used in the experiments. In addition, the standard IIOP protocol suite was used for interoperable communication. The management broker was based on Orbix 2.2 and the OSIMIS-4.0 Generic Managed System (GMS). The experiments were performed on the Sun Sparc 5 / Sparc 20 pair of workstations running Solaris. It was not possible to repeat them on the 486 PCs since Orbix does not work on the Linux version of UNIX.

4.5.1 Program Size

Every network node that can host CORBA programs needs to run the local part of the ORB. This is equivalent to the ODP concept of nucleus and in Orbix is implemented by a daemon process, the *orbixd*. The latter is around <u>2700 Kb</u> at run time. There is no equivalent daemon in the OSI-SM environment since the upper layer stack is directly linked with the applications and the agent plays to some extent the role of the ORB.

We will examine first the size of a CORBA server process containing managed objects with native IDL interfaces. A CORBA server is equivalent to the ODP concept of capsule. We will consider the *i_uxObj* interface that was depicted the in Code 4-1 caption and is fully specified in Appendix C. The only difference is that there is no i_ManagedObject interface i.e. the inheritance hierarchy is *i_uxObj -> i_top -> CORBA::Object*. This is exactly equivalent the uxObj GDMO class, so that we can draw relevant comparisons. The factory for this interface is *i_uxObjFactory*, as it was specified in the Code 4-3 caption.

The size of the server process that contains one instance of the *i_uxObjFactory* and the *i_uxObj* interfaces, with the attribute values for the latter as specified in the Code 3-12 caption of Chapter 3, is <u>2640 Kb</u> at run time. The equivalent size of an OSI-SM agent with the same functionality was 1340 Kb but the OSI agent supports also scoping, filtering, event reporting and logging. In addition, the orbixd is needed on that node, which increases the overall memory requirements. The size of the smallest possible server is <u>2520 Kb</u>, measured for a server with an i_echo interface that supports a simple echo operation.

The amount of code linked in for the *i_uxObjFactory* and the *i_uxObj* interfaces is about <u>80 Kb</u> at run time while the equivalent amount for the uxObj class was around 20 Kb. In the latter case there is no need for a specific factory class since the agent plays the role of a generic factory.

The data overhead of a *i_uxObj* instance with the attribute values as above is exactly <u>896 bytes</u>. The equivalent overhead of a GDMO *uxObj* instance was 420 bytes. In the latter case though the object is part of the MIT and can evaluate filters; this functionality incurs additional memory overhead due to the required instance variables.

The size of the client program that invokes an echo action on the i_uxObj interface is <u>2540 Kb</u> at run time. The equivalent size of the OSI-SM manager program was 1060 Kb.

Finally, these experiments were performed without the CORBA name server: the server program printed out the i_uxObj reference to a file and the client program read it from there. When the name server is used, the size of both the client and server programs are slightly bigger since they would need to link in the name server's interface. The size of the name server that handles a naming graph of modest size is about 2800 Kb at run time.

All the above measurements concern native CORBA managed objects. If the management broker approach is followed, the size of the *i_CMISBroker* server object is comparable to the i_uxObj server object. The overhead of managed object instances in this case is the same as in the OSI-SM approach, since they are held internally as OSIMIS managed objects.

Description	Size (Kb)
ORB daemon	2700
Server object size (without name server access)	2640
Client object size (without name server access)	2540
Overhead of the i_uxObj / i_uxObjFactory interfaces	80
Overhead of the uxObj instance as in Code 3-12	0.896
Small Name Server size	2800

Table 4-2 Summary of the CORBA Memory Overheads

Table 4-2 presents the summary of the program size overheads for the native CORBA based approach. In summary, the overheads are at least double of those of OSI-SM which were presented in Table 3-7 of Chapter 3. The key problem is the size of server object: the smallest possible server is around 2.5 Mb at run-time. The initial idea behind ODP was to be able to distribute objects as much as possible so that they can benefit from the various transparencies independently, without being clustered together. This type of distribution requires many servers to host those objects but the size of a server seems to be prohibitive for fine grain distribution. While this program size is acceptable for the ORB daemon, it is too big for server objects.

On the other hand, the memory overhead of an object instance within a server process is relatively reasonable, though twice the size of that of OSIMIS. The key reason for the bigger overhead compared to OSI-SM is the size of the IIOP Inter-Operable Reference (IOR), which is at least 200 bytes. [Whit97] accepts this as an issue that affects CORBA scalability and proposes compositional models of *entities* and *warehouses* so that the number of references are reduced. The management broker approach is in fact such a compositional model.

4.5.2 Response Times

We will now examine the response times for the *echo* operation of the *i_uxObj* interface. Before the operation takes place, the client object needs to establish an interface reference to the object in server role. This is typically done by resolving a name to an interface reference through the naming server, as already discussed. After that, the client performs directly the operation without the need to explicitly establish a connection to the server. Since IIOP uses the Internet TCP/IP, a TCP connection will need to be established by the infrastructure but this is totally transparent to the object. Since a connection is implicitly established, an increased response time is experienced the first time a client accesses a server object. We can thus calculate the response time for establishing an underlying connection but we cannot calculate the response time for releasing it.

Performing an echo operation for the "hello" string results in 5 msecs response time. The equivalent operation on the uxObj instance using the Q_3 interface was 11 msecs. Note that the figure for the latter presented in the Table 3-9 of Chapter 3 was 12 msecs but the operation was performed asynchronously: the synchronous operation is about 8% faster as explained in Chapter 3 or 11 msecs. This is a significant performance advantage compared to OSIMIS i.e. 45% of its access time. The explanation for this lies mostly in the supporting protocol stacks and is according to expectations. The CMIS m-action request and response messages contain other parameters in addition to the action type and echoed string, e.g. the object class, name, time etc., and their encoding and decoding causes additional overhead.

Increasing gradually the size of the echoed string results in an almost linear increase of the response time. The additional overhead is around 0.3 msecs per 100 bytes or another 0.06% of the overall response time. The equivalent figure for OSI-SM is 0.21 msecs per 100 bytes. The fact the OSI-SM figure is smaller than the CORBA one is unexpected. A detailed comparison between the CORBA encoding rules and the BER is required to investigate this further.

The first time the operation is performed it takes 17 msecs because the underlying TCP connection is established. This implies that connection establishment takes roughly 17-5 = 12 msecs. The equivalent time for establishing a CMIS association in OSIMIS is 39 msecs. This is again very favourable for CORBA i.e. roughly 30% of the OSI-SM association establishment time. It can be easily explained because of the various negotiations that take place in the OSI upper layers, involving a number of packet exchanges as described in Chapter 3.

We will now examine the same echo operation performed indirectly to the managed object through a synchronous operation to the *i_CMISBroker* interface. The *action* method signature has been depicted in the Code 4-5 caption and is equivalent to a "synchronous version" of the

CMIS m-action primitive: the object name and class are passed in addition to the action type and action information. This operation takes 8 msecs in comparison to 11 msecs through the Q_3 interface i.e. 72% of the Q_3 access time. This operation is more expensive than accessing directly the i_uxObj interface for two reasons: first, additional parameters are passed across a similar fashion to CMIS/P; and second, a mapping is necessary between the IDL C++ data types and the equivalent OSIMIS ASN.1 C++ types.

Performing the same operation asynchronously takes 9.2 msecs in comparison to 12 msecs through the Q_3 interface, which amounts to 77% of the Q_3 access time. The signatures for the "one way" *actionAsync* and *actionResult* operations have been depicted in the Code 4-5 and Code 4-7 captions respectively. The additional overhead in this case is again because of two reasons: first, additional parameters are passed across in both the request and the response primitives; and second, asynchronous operations are always slightly slower than synchronous ones as it was discovered in Chapter 3. It should be finally stated that performing an operation asynchronously is exactly equivalent to the corresponding CMIS operation: the same parameters are passed across in the request and response primitives.

[Schmi97] has also addressed CORBA performance. The relevant experiments in that case were done over ATM using the most powerful Sun Sparc stations. Figures as low as 2 msecs for a parameterless operation are reported. [Park96] has also compared a CORBA implementation with OSIMIS and concludes that CORBA is around 40% faster, which is not far from the figures presented above.

In summary, CORBA performance is better than OSI-SM when measuring the Orbix and OSIMIS implementations respectively. This is particularly true when managed objects with direct IDL interfaces are used. When a CMIS-like approach through the management broker is used, the performance difference is smaller. While these performance differences are not dramatic, there is ongoing work towards real-time ORBs that will improve CORBA performance further. On the other hand, the proximity of the Orbix and OSIMIS performance figures verifies the good performance of OSIMIS and OSI-SM in general in this comparative context.

4.5.3 Packet Sizes

We finally consider the sizes of management packets, which include the payload over the TCP as explained in section 3.8.1 of Chapter 3.

Establishing a connection between objects in different network nodes does not involve any additional procedures to TCP connection establishment. This means that only two TCP packets are exchanged with zero payload. In contrast, OSI-SM requires 4 additional packets which contain 310 bytes of data as it was shown in Table 3-10. The same is true for connection release: OSI-SM requires 3 additional packets which contain 64 bytes of data as it was shown in Table 3-11. In summary, connection establishment and release in the CORBA IIOP relies on the underlying TCP procedures. As such, it generates much less traffic that OSI-SM.

Performing an echo operation with an empty string on the i_uxObj interface results in a request packet of 101 bytes and a response packet of 29 bytes. Performing the same operation through the synchronous management broker interface results in request and response packets of 177 and 37 bytes respectively. Finally, performing the same operation through the asynchronous management broker interface results in request and response packets of 409 and 97 bytes respectively. This last operation is exactly equivalent to the CMIS/P one for which the relevant sizes are 72 and 88 bytes respectively as it was shown in Table 3-12.

The interesting result here is that CORBA seems to generates a relatively large amount of traffic. The payload for the simplest operation is 101/29 bytes and these packet sizes increase significantly with additional argument and result parameters. This increase becomes much bigger when IORs are passed across, as experienced in the asynchronous operation in which the request packet size "shot up" to 409 bytes. This is expected because of the size of IORs. It appears though that the CORBA encoding rules are at least as verbose as the BER [X209].

These measurements also verify the fact the size OSI-SM operation packets is relatively modest, despite the number of parameters required by CMIS/P.

4.6 Summary

4.6.1 Overview of this Chapter

In this chapter we presented first an analysis of ODP as the theoretical framework for distributed systems. In this analysis, we discussed OSI-SM and TMN from an ODP perspective and presented briefly two different approaches for mapping OSI-SM to ODP. We then presented ANSA, the OSF DCE and OMG CORBA as three generations of ODP-influenced technologies. An analysis showed that both ANSA and the OSF DCE failed to satisfy essential TMN requirements i.e. object-orientation and dynamic operation without pre-compiled knowledge. On the other hand, OMG CORBA seems to be the first ODP-influenced technology that has come of age, satisfying most of the distributed framework requirements identified in Chapter 3. A drawback is that its IDL abstract language does not support polymorphic distributed operations.

Given the emergence of CORBA as the ubiquitous object technology for open distributed systems, we examined in detail how it can be used as the base technology for the TMN. We started from an ODP-influenced approach in which trading is used instead of hierarchical naming for object discovery and demonstrated the problems with this approach in management environments with large sizes of distributed objects. We subsequently presented a minimal approach which retains the TMN hierarchical naming and containment relationships but does not support scoping, filtering, multiple object access and fine-grain event reporting and logging. A key aspect is that only few objects in each "agent" domain need to export their names to the name server. This avoids scalability problems in TMN environments where network elements and operations systems may contain 10's of thousands of objects.

We then added multiple attribute access and filtering to the managed objects and explained how CMIS-like multiple object access can be supported through the management broker. This was done in an incremental fashion, without mixing CMIS-like access aspects with the managed object interfaces. We finally exploited the filtering capability of managed objects in order to add EFD-based fine-grain event reporting. The proposed architecture retains the advantages of OSI-SM but with the following drawbacks: support for filtering and knowledge about the attributes of a particular object need to be hand-coded i.e. they cannot be automatically supported by IDL compilers. OMG may re-consider its attribute model in the future and add expressiveness similar to GDMO; this will solve these problems. The management broker approach was validated through implementation and was used in project trials.

Finally, a brief performance analysis and evaluation of the CORBA-based framework was attempted with the main target to compare the Orbix implementation of CORBA to the OSIMIS implementation of OSI-SM. The results have shown CORBA to exhibit faster access times than OSI-SM, in particular when accessing managed objects directly. Access through the management broker is slightly slower but still faster than OSI-SM. The penalty for this increased performance seems to be the memory requirements: the size of a server process is too big and this discourages fine-grain partitioning of objects into servers for distribution purposes. In addition, the memory overhead of an object instance is at least twice that of OSI-SM. Finally, the traffic incurred seems to be at best similar to OSI-SM and much bigger when object references are communicated.

In summary, CORBA seems a very promising technology which can form the basis of future TMN systems. Its value compared to OSI-SM is not so much the slightly better performance but the fact that it may become the ubiquitous technology for future heterogeneous distributed systems. Its use in both management and service control environments will result in economies of scale and allow for easier integration of new managed resources.

We should finally answer the question of what is the architectural impact to the TMN if CORBA is adopted. The answer is that there is no impact at all. The TMN architecture and methodologies will remain exactly as presented in Chapter 2. Interface specifications will be produced in GDMO, following possibly guidelines which will guarantee that generic translation to IDL is possible. In addition, the Q_3 interface profiles will be modified, including CORBA protocols as valid choices for TMN interfaces. The use of CMIS/P-GDMO or GIOP-IDL will become an engineering issue for implementing the same specifications. It should be added though that mappings of GIOP over lower level protocols other than TCP/IP are necessary in order to meet the needs of telecommunications environments. A GIOP mapping to SS7 is currently being addressed by OMG. Finally, an additional benefit of using CORBA is that TMN OS components could be distributed across different network nodes.

In summary, CORBA can be seen as an object-oriented version of ROSE which can support any transaction-based application layer protocols. It was shown in this chapter how CMISE can be supported through the CORBA-based management broker. The addition of stream interfaces and quality of service parameters [Hand95][Kits95] will make CORBA the ubiquitous infrastructure that will be able to support any application layer protocols, involving stream transfers as well as operational transactions.

4.6.2 Research Contribution

The first part of this chapter introduced the ODP framework and associated technologies in the context of telecommunications management, concentrating mostly on OMG CORBA. Despite the fact that the relevant sections served mainly as state-of-the-art presentations, they were presented from the point of view of suitability for telecommunications management and, as such, they constitute to some extent research contributions in their own right. In this section we state clearly the research contributions in this chapter.

- The suggestion of two different approaches for describing OSI-SM / TMN in ODP terms in section 4.2.5. The second approach, in which an OSI-SM/TMN agent becomes a computational object, was elaborated further through the Management Broker approach.
- The analysis of the properties of ANSA and OSF DCE as candidate technologies for telecommunications management in sections 4.3.1 and 4.3.2 respectively. The conclusion that they failed to satisfy important requirements.
- The analysis of the properties of OMG CORBA as a candidate technology for telecommunications management in section 4.3.3. The identification of deficiencies such as true polymorphic operation, lack of built-in support for asynchronous operations and not expressive enough attribute and event models. On the other hand, the finding that CORBA satisfies most of the desired properties of distributed object frameworks and can be used as base technology for the TMN.
- The discussion of the JIDM mapping rules between GDMO and CORBA IDL [JIDM95] in section 4.4.2 and the identification of problematic aspects regarding the mapping of attributes and conditional packages.
- The analysis of how the OSI-SM / TMN model could be mapped to CORBA in a pure ODP fashion which uses trading and avoids hierarchical naming schemes in section 4.4.3.1. This approach needs federated trading which is very difficult to provide in non-hierarchical service spaces. In addition, the OSI-SM / TMN access aspects are lost and this results in increased management traffic.
- A similar analysis which retains the OSI-SM / TMN hierarchical naming and containment relationships for managed objects in section 4.4.3.2 and the proposed "minimal" architecture in section 4.4.3.4. This can be easily provided with the current state of CORBA technology and is unrelated to CMIS as an access method.

- The analysis of how the previous minimal architecture can be enhanced with multiple object access through scoping and filtering and fine-grain event dissemination in section 4.4.4. The presentation of the Management Broker approach which adds facilities other than filtering in an orthogonal fashion to managed objects. The complete specification, implementation and validation of the management broker approach.
- The presentation of an evolutionary approach towards a CORBA-based environment in section 4.4.4.4. In this, management brokers can be implemented in various stages: first as gateways, then as dual agents, subsequently as plain CORBA-based agents and finally as brokers for CORBA-based MOs with native IDL interfaces.
- Finally, the performance analysis and evaluation of the CORBA-based framework in section 4.5 and its comparison to the OSI-SM one. The main findings were that CORBA servers and objects are relatively big, packet sizes are not small but the performance is very good, especially when accessing MOs directly, through their native IDL interfaces.

It should be finally mentioned that the research contributions in this chapter are the author's alone. The starting point for this research work has been the work of the JIDM group on the comparison of the OSI-SM and OMG CORBA object models and the guidelines for translating OSI-SM GDMO definitions to CORBA IDL. The author would like to thank T. Rutt of Lucent Technologies, the editor of the comparison of the object models document [Rutt94]; S. Mazumdar, also of Lucent Technologies, the editor of the GDMO/ASN.1 to IDL translation document [JIDM95]; and T. Tin of UCL for implementing the gateway version of the management broker.

5. Summary and Conclusions

In this final chapter we bring together the work described in the previous chapters of this thesis. Section 5.1 explains the significance of this thesis in terms of the contributions to standards bodies and to the wider telecommunications management community. It also summarises the main findings. Section 5.2 identifies areas in which this work could be developed further.

5.1 The Contribution and Main Findings of This Thesis

The key contribution of this thesis was to explain and simplify the TMN architectural framework, to demonstrate its feasibility, implementability and performance when OSI-SM is used as the base technology and to show how distributed object technologies such as OMG CORBA can be used to replace OSI-SM in the future.

Given the fact that the relevant research work was undertaken over a long period of time, a number of the findings have already been published in research papers and contributed to international standards bodies and the wider telecommunications management community. The proposal for the integration of the OSI Directory in the TMN has found its way to the TMN architectural framework [M3010]. The precise details of using the OSI Directory for shared management knowledge and location transparency have been contributed to the Management Knowledge SMF [X750]. It was also proposed to the ITU-T Study Group 4 to remove the mediation function / q_x reference point and the f reference point from the TMN architecture. The proposal for removing the MF / q_x has been adopted in the latest 1997 draft of the TMN framework [M3010]. The proposal for not addressing the f reference point has not yet been adopted but almost no work exists to date behind its standardisation.

The above findings are of architectural nature and have been backed up by design, implementation and experimental results with real TMN platform infrastructure and prototype TMN systems. These experiments have shown that full scale OSI-SM / TMN technology is feasible, performant and economical to provide both in terms of computing resources and required development time. In addition to the architectural contributions to standards bodies, the OSIMIS TMN platform was made available to selected companies and research institutions as early as in 1991 i.e. version 2.95. Subsequent versions were made publicly available to the wider telecommunications management community i.e. version 3.0 [Pav93a][Pav93b] and version 4.0 [Pav95b]. The relevant concepts, models, object-oriented design aspects and parts of the actual software itself were subsequently used in a number of commercial TMN offerings. Finally, the relevant models and APIs were provided as input to the NMF TMN/C++ API group at the end of 1995. Their recently proposed solution [Chat97] bears a lot of similarities with the concepts and object-oriented models detailed in Chapter 3 of this thesis.

The research work related to the use of CORBA as the base technology for the TMN was undertaken during the last two years, i.e. in 1996 and 1997. Though some of the results have already been published [Pav97b][Pav97d], the complete approach as presented in Chapter 4 has not yet been made publicly available. Given the growing interest in the use of distributed object technologies in TMN, the author intends to propose the relevant architecture and specification to standards bodies. This work will be specifically proposed to the OMG Telecommunications Special Interest Group; to the ITU-T SG 4 which addresses TMN evolution and its de-coupling from OSI-SM [M3010]; and to the evolving Open Distributed Management Architecture [X703] as an addendum on the potential use of CORBA.

Given the fact there is not yet a complete solution in the research community for replacing OSI-SM with CORBA in TMN environments, the author's proposal constitutes the major contribution of this thesis. It also justifies the fact that the thesis was submitted two years later than initially intended. The rest of the contributions on the TMN architectural framework and its validation through the design and implementation of the OSIMIS platform are equally important. It should be noted that it would have been impossible to propose the complete solution for the use of CORBA in TMN environments without the design and implementation experience from the OSI-SM based approach.

The research contributions of this thesis were summarised at the end of chapters 2, 3 and 4. We re-iterate through the main findings below.

The findings related to the TMN architecture are the following:

- OSI-SM is suitable technology for TMN systems because it satisfies a number of key TMN requirements: object-oriented information modelling aspects; intelligent information retrieval facilities; and fine-grain control event of event dissemination.
- OSI-SM does not support location transparency but it can be combined with the OSI Directory which can support name resolution for OSI-SM management applications. The key advantage of the OSI Directory is its federated nature which can support wide-area geographic distribution.
- There is no point in endorsing "less capable" TMN network elements through the existence of the lightweight Q_x interface: there are far too many possibilities for different Q_x interfaces which need specialised mediation devices. In addition, the cost of the full-scale Q_3 interfaces is small as it was explained in Chapter 3 of this thesis.
- There is no need for lightweight TMN F interfaces to drive WS applications: the overhead of full-scale Q_3 interfaces can be very easily accommodated by inexpensive laptop and desktop computers, which can provide views of telecommunications activity from diverse geographic locations. Given the prominence of the WWW as the ubiquitous network GUI, there is the possibility of encapsulating CMIS/P or CORBA IIOP messages in HTTP. There are various different ways of how this can be done and, as such, the TMN F interface should not be standardised.
- Given the previous findings, the TMN becomes a fairly simple architectural framework in which Q_3 is the intra-TMN and X the inter-TMN interface. The key TMN characteristic becomes its layered hierarchical nature (element, network, service and business management) and the fact that specifications are produced only for inter-layer and inter-TMN interfaces, allowing flexibility for different realisations.

The findings related to the object-oriented realisation of the OSI-SM / TMN framework are the following:

- CMIS/P is a fairly straightforward protocol which needs only a connection-oriented reliable transport service and a presentation facility. Lightweight versions are also possible based on string encodings. A CMIS API can be used as a building block for higher-level infrastructures. These can hide its details and complexity and provide a development environment for managed and managing objects, in a similar fashion to emerging distributed object technologies such as OMG CORBA. A key ingredient for such infrastructures is an object-oriented polymorphic ASN.1 API.
- There exist two types of object-oriented manager infrastructures: those that model whole agents (the Remote MIB) and those that model individual managed objects (the Shadow MIB). Shadow objects may be generic, weakly-typed or specific, strongly-typed. In the latter case they may be enriched with behaviour which exploits the semantics of particular managed object classes. The RMIB and the strongly-typed SMIB are the best approaches. The SMIB approach entails a manager mapping of GDMO to O-O programming languages. It is also possible to map the RMIB and SMIB approaches onto scripting languages such as Tcl or, better, Java.
- Managed objects may be shielded from underlying protocol and service access details through generic object-oriented manager infrastructure which maps GDMO to O-O programming languages. Polymorphic methods may be used for the enrichment of automatically produced "stub" managed objects with behaviour. Aspects such as name resolution, scoping, filtering, atomicity, persistence, access control and event dissemination can be provided in a transparent fashion to managed object logic and behaviour. Finally, SMFs model resource independent generic aspects and can be supported in a fully re-usable fashion.
- TMN applications may be organised in a single or multi-threaded fashion. In the former case, asynchronous APIs should be used for remote invocations while sequential activities should be "conscious" about their duration. Long-lasting activities should be delegated to other operating system processes so that the "core" application remains responsive. The advent of kernel-based multithreading and relevant support in modern operating systems suggests that multi-threading is the way forward. On the other hand, concurrency control remains an issue that requires special support and care in complex applications.

- A detailed performance analysis of the OSIMIS platform, which validated the object-oriented ideas for the realisation of the OSI-SM / TMN framework, showed that full-scale TMN technology is relatively inexpensive in terms of computing resources and performant. The infrastructure overhead for a TMN OS is about 1.5 Mb at run-time, the data overhead for a simple MO is 0.5 Kb while an echo action takes around 10-11 msecs on a LAN using a typical pair of UNIX workstations. Packet sizes are reasonable for management operations (in the order of 100 bytes / packet) but a number of packets are necessary for establishing a management association. The major performance overhead is due to the OSI protocol stack rather than the object-oriented application framework.

Finally, the findings related to the use of OMG CORBA as the base technology for the TMN are the following:

- When considering OSI-SM in the ODP framework, it is possible to consider an agent application as an ODP computational object which contains managed objects as ODP information objects. The whole agent becomes an engineering object that supports a CMIS-like computational interface. Managed objects become plain engineering objects without computational interfaces, accessed in a local manner by the agent. Until now, the OSI-SM to ODP mappings in the literature considered always managed objects as separate computational objects.

- ANSA and the OSF DCE failed to satisfy the following important requirements for telecommunications management: an object-oriented information and computational model with inheritance; support for the *any* type in their IDL so that dynamic, weakly-typed interfaces are possible; support for recursive structures so that CMIS-like filters can be expressed; and support for a dynamic invocation interface which is necessary for generic, semantic-free applications such as MIB browsers. In addition, both ANSA and DCE supported only C language APIs.

- CORBA is an ODP-influenced technology that has come of age. It supports interface inheritance, the any type, a dynamic invocation interface and multiple O-O programming language bindings. It still lacks though: full support for polymorphism through the redefinition of operations in derived interfaces; fine-grain control of events based on filtering; and asynchronous APIs.

- The JIDM GDMO to CORBA IDL mappings can be used as the basis for a native CORBA-based architecture. The problem is that the notion of attributes is lost which makes difficult to support "get all" attribute operations. In addition, in IDL attributes there is no automatic support for pretty-printing, comparison and subsequently filtering.
- According to pure ODP principles, managed objects should not exhibit hierarchical names and references to them should be discovered through the trader. This approach requires federated trading for non-hierarchical service spaces, which is as yet an unsolved problem.
- A more pragmatic approach is to retain the TMN hierarchical naming principles, organise managed objects in "agent" domains according to containment relationships and use the CORBA name services to get access to the "virtual MIT" root object. Managed objects could resolve names of subordinate objects to references. This is a minimal approach, with no multiple attribute access, scoping, filtering, multiple object access and fine-grain event management.
- The missing OSI-SM functionality may be added through an "attribute repository" that can be used to support multiple attribute access and filtering, the last through hand-coded comparison methods. Support for filtering makes possible to provide OSI-SM-like event management facilities through an Event Processing object and EFD / log objects. Multiple object discovery and access through scoping and filtering may be provided by a Management Broker object, in an orthogonal fashion to managed objects.
- Migration from OSI-SM to a CORBA-based TMN can be handled gracefully by implementing management brokers at various stages: first as CORBA to OSI-SM gateways, then as dual CORBA and OSI-SM agents, subsequently as CORBA agents and finally as proper brokers for MOs with native CORBA IDL interfaces. This approach will retain investment on OSI-SM-based TMN technology for as long as necessary.
- A performance evaluation of the CORBA-based framework using a commercial CORBA platform showed very good access times, i.e. 4-5 msecs for an echo action through a native IDL interface on a LAN using a typical pair of workstations. On the other hand, the data overhead for a managed object instance and the smallest application size were at least double those of OSIMIS. Finally, packet sizes are comparable to the OSI-SM ones but become much bigger when interoperable object references are communicated.

- The advantage of using CORBA is not so much its slightly better performance but its better distribution paradigm, the software portability and the possibility that it may become the ubiquitous distributed object technology. In the latter case, its use in the TMN may result in economies of scale.
- The use of CORBA does not require any changes to the TMN architectural framework. CORBA protocols will be endorsed as valid options for the Q_3 and X interfaces. Information model specifications will still be in GDMO while the use of CORBA or OSI-SM will become an engineering decision. Interoperability between the two will be supported through gateways in both directions.

It is finally worth mentioning that if CORBA with C++ language bindings was available around 1992-93, the author would have used it as the basis for an object-oriented TMN platform. The fact that ODP-influenced technology at the time had a number of limitations (i.e. ANSA, DCE) led the author to the design and implementation of the OSI-SM based TMN platform. This was essentially a "management-oriented DPE" and the advent of technologies like CORBA justified this initial vision and direction. The existence of an OSI-SM based TMN platform allowed research and experimentation with TMN systems in the first half of the nineties, which would have otherwise been impossible.

5.2 Potential Future Work

There are various ways in which this work can be taken further, we summarise potential future directions below.

- The Lightweight CMIP approach [Pav95c] was never more than a specification document. The recent success of the equivalent Lightweight Directory Access Protocol [LDAP], on which LCMIP is based, and the management needs of mobile network environments suggest that this approach could be taken further. An implementation and subsequent detailed performance measurements and comparison to the full Q_3 approach is necessary in order to quantify potential advantages.

- The OSI-SM performance measurements in section 3.8 of Chapter 3 represent only a first approach towards a performance analysis of OSI upper layer protocols. The increasing popularity of the current TMN approach in which OSI-SM is used as the base technology suggests that a more detailed analysis would be welcome. This could highlight those aspects of the upper layer stack that cause the overhead and propose subsequent optimisations. The existence of the full source code of OSIMIS / ISODE makes possible to undertake such a performance study.

- In the same fashion, the performance analysis of the CORBA-based framework in section 4.5 of Chapter 5 only scratched the surface of the problem. A detailed performance analysis is necessary in order to attribute the overheads to various parts of the system and propose optimisations. This work though requires access to the source code of a CORBA platform.

- The specific, strongly-typed Shadow MIB approach is pertinent to OSI-SM based TMN platforms, e.g. OSIMIS, NMF TMN/C++, and does not exist in CORBA. Its key advantage is that shadow objects can be enhanced with class-specific behaviour for caching, adaptive polling, etc. It would be interesting to see how such an approach can be provided over the CORBA strongly-typed client API. This work will require access to an IDL compiler and may result in changing the IDL to O-O language mappings.

- Federated naming services in CORBA can be provided in a similar fashion to the OSI Directory due to the hierarchical nature of the name space. Federated trading and federated event services though are much more difficult to provide. Research is necessary to address the optimal provision of such federated services.

- Management is impossible without security. While management issues have been addressed in OSI-SM environments, it is interesting to investigate how similar facilities can be supported in CORBA-based environments. A particularly interesting issue is the provision of access control. This can be relatively easily provided for agent-administered managed objects but becomes a much more difficult proposition for managed objects with native CORBA IDL interfaces. A related issue is the scalability and performance of the relevant solution.
- The advent of languages like Java that can support code mobility opens up new possibilities for management. The impact of this new paradigm to TMN is a whole new area of interesting research, pointing to "programmable" network elements with very simple native interfaces and an environment that can host mobile software entities.
- The TMN was developed as a framework addressing only the operation, administration and maintenance activities of telecommunications networks. It has become clear though that a more general framework is necessary for the integration of all distributed telecommunications software, including both management and service control. The evolution of the TMN together with other related frameworks such as IN and TINA towards a unifying framework is an interesting research issue.
- One of the main research interests of the author lies in performance and quality of service management for multi-service telecommunications networks. The existence of environments like the one presented in this thesis will make possible to address this problem with experimental management systems, in addition to analytical studies and simulation.

Appendices

Appendix A: Work Based on the Proposed TMN Development Environment

The object-oriented software architecture for the TMN development environment proposed in this thesis was validated through the design and implementation of the OSIMIS platform. Since the latter is a generic development environment, the validation of the proposed architecture and the software platform itself was ultimately achieved through the design, implementation and deployment of a number of TMN systems in various research projects. In fact, the existence of such a development environment both stimulated and enabled additional research in this area. In this appendix we present briefly research and development work based on the proposed development environment.

A.1 Research and Development Work Involving the Author

The very first application developed using OSIMIS was an agent for managing the ISO/ITU-T Transport Protocol [TPMIB]. This was developed by the author and was a re-engineering of the same application that had been previously developed by S. Walton of UCL. This particular application validated the early version of the OSIMIS Generic Managed System. It used to be part of OSIMIS releases until version 3.3 and served as an example of relatively sophisticated use of the agent part of the infrastructure.

The next application developed with OSIMIS was the generic MIB browser [Pav92a], which has since become part of OSIMIS. This was an important manager application that highlighted necessary support aspects for generic manager applications. It also highlighted the need for high-level manager infrastructure and led to the specification and embryonic implementation of the OSIMIS RMIB. The MIB browser was developed by J. Cowan of UCL.

During the summer of 1991, a MSc student group project produced a first version of an agent for managing the OSI version of the Internet SNMP MIB-II [OIM]. This was taken further after one of the students stayed at UCL as a research associate after his MSc. This application stretched aspects of the agent infrastructure since it involved extensive interaction with the UNIX kernel. It was developed by S. Bhatti and was included in OSIMIS versions prior to 4.0.

During the summer of 1992, an MSc student group project produced a "proxy" implementation of the OSI version of the Internet SNMP MIB-II [OIM]. This was based on a non-generic model but highlighted the possibilities for a generic CMIS/P-SNMP proxy agent. One of the students, K. McCarthy, stayed at UCL as a research associate after his MSc and was involved in the research, design and implementation of a generic proxy agent [McCar95]. This is known as the Internet Q-Adapter (IQA) and is part of the OSIMIS 4.0 distribution. J. Reilly of VTT designed and implemented the supporting SNMP SMI to GDMO MIB translator.

During the summer of 1994, an MSc student group project produced a comprehensive local area network monitoring tool. This reproduced most of the functionality of the SNMP Remote Monitoring MIB [RMON] by re-using the OSI SMFs supported by OSIMIS. It demonstrated the use of generic infrastructure to simplify the task of producing complex applications such as a RMON agent.

A PhD student, N. Vassila, investigated the applicability of the "management by delegation" paradigm [Yemi91] in TMN environments. She used the Tcl scripting language to introduce Active Managed Objects (AMOs) in the OSIMIS agent framework. Such MOs can be dynamically downloaded to agent applications [Vass95][Vass97].

The first hybrid manager-agent TMN OSs were developed in the RACE NEMESYS project as part of a TMN system for ATM Quality of Service management. An Element Manager OS and a Service Manager OS were developed, managing a simulated ATM network through a relevant Q-Adapter. The architecture of this system is described in [Pav91b] while its modelling, design and the implementation experiences are described in [Pav92b]. This was the first hierarchical TMN system with complete Q_3 interfaces. The Q-Adapter verified the scalability of the agent infrastructure since it contained more than 10000 managed objects for the simulated ATM network that was used in the experiments.

OSIMIS was used in a large scale in the RACE ICM project. The project's TMN system is described in detail in [Gri95][Gri96b][ICM] and provided ATM Virtual Path Connection and Routing Management (VPCRM) services and also ATM-based Virtual Private Network (VPN) services. This was the most complex TMN system built at the time, comprising twelve different types of TMN Operations Systems (OSs). These could be instantiated in different domains, providing end-to-end VPN services and intra-domain VPCRM services. The system operated over both real and simulated networks. It was developed mostly by researchers in different companies and research institutions around Europe who had little or no exposure to network programming. It demonstrated the suitability of the proposed framework for the rapid development of sophisticated TMN systems.

Various research topics were addressed by the ICM project as part of its TMN activities. [Georg95] proposes an approach to realising management services for performance verification in multi-service ATM networks. These have a minimal impact on the managed network by delegating performance management activities to the network elements. This is achieved through the use of the OSI Metric Monitoring [X739], Summarisation [X738] and the Intelligent Monitoring Function (IMF) [Pav96c]. The latter was devised, designed and developed in the ICM project. It combines the power and expressiveness of the metric monitoring and summarisation functions and allows a manager application to form complex expressions of monitored attributes and delegate them to the agent. The IMF support objects were implemented by G. Mykoniatis of the National Technical University of Athens (NTUA).

An important aspect of the ICM project was the demonstration of TMN security services, mainly inter-domain across the X interface but also intra-domain. A lightweight approach was adopted for the authentication, stream-integrity and confidentiality services, based on symmetric, secret-key based encryption and signing functions. The security architecture, transformations and programmatic access aspects are described in [Bhat96]. S. Bhatti and G. Knight of UCL were the major architects of the solution while K. McCarthy implemented a major part of it. In addition, the author designed and implemented the object-based access control function together with T. Tin of UCL. The lightweight security and access control functions are part of the OSIMIS 4.1 version.

TINA applications based on the CORBA-based version of OSIMIS were used in the ACTS VITAL and REFORM projects for Resource Configuration Management (RCM), as described in [Pav97b]. These were TMN-like applications holding network topology information through managed objects with various relationships, according to the TINA Network Resource Information Model (NRIM) [NRIM]. Access to those was provided through the CORBA-based "Q_3" management broker interface that was described in Chapter 4. These applications were developed by the author and T. Tin of UCL.

A.2 Research and Development Work by Others

OSIMIS was used in a number of other RACE, ACTS and ESPRIT projects, in addition to the NEMESYS, ICM, VITAL and REFORM projects mentioned above.

In the RACE PREPARE project it was used in addition to a number of commercial platforms for developing TMN applications, as described in [Lewis95]. Parts of it were also used to support the PREPARE Inter-Domain Management Information Service (IDMIS) [Bjer94] platform. It is also currently being used in the ACTS PROSPECT project, which is the continuation of PREPARE. PROSPECT uses the Tcl-RMIB infrastructure [Tirop97]. OSIMIS is also being used in the ACTS MISA project.

In the RACE TOMQAT project it was used for developing quality of service management applications [Lioup94]. In the RACE BAF project it was used for developing Q-Adapter TMN applications. It is also currently being used in the ACTS BONAPARTE project, which is the continuation of BAF. There it is used for developing TMN NE agents for ATM switching equipment [I751].

In the ESPRIT PROOF project it was used to develop an OSI-based management system for a primary rate ISDN to Internet IP gateway. In the ESPRIT MIDAS project it was used to develop applications for managing OSI X.500 directory systems and X.400 mail systems. It was also used as a vehicle to do research in and develop asymmetric public-key based security services as described in [Bhat95]. In the ESPRIT IDSM project it was used to implement intelligent monitoring policies.

OSIMIS has also been used by researchers all around the world in various research projects. Related research work is presented below, based mainly on publications in the IFIP/IEEE Integrated Management (IM) Symposium, the IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM), the Intelligence in Services & Networks (IS&N) conference and the Journal of Network and Systems Management (JNSM).

[Filip93] describes an experimental agent structure for migrating IBM's SNA-based network management aspects to OSI-SM environments. The agent translates between SNA management information and protocols to GDMO and CMIS/P and was based on IBM's adaptation of an early version of OSIMIS.

[Jord93] describes a model and associated prototype for alarm correlation, which tries to identify the faulty resource with high probability. The prototype was developed at IBM ENC and was based on IBM's adaptation of an early version of OSIMIS.

[Ditt95] describes the ANDROMEDA platform (Analysis, Development and Provision of a Management Environment for Distributed Applications), which was developed by GMD Fokus as part of their BERKOM project. It is a collection of tools and environments which reuses the OSIMIS agent infrastructure and MSAP manager API together with the ISODE QUIPU DSA [QUIPU]. It also provides a uniform access interface to both management and directory information known as IDMIS - Inter-Domain Management Information Service [Bjer94].

[Sartz95] describes the design and implementation of a meta-management system for managing the TMN itself. This includes a system for placing the various TMN applications in particular network nodes, monitoring their activity and potentially relocating them for performance of fault reasons. The relevant prototype system was based on OSIMIS.

[Perr95] proposes a model and an associated prototype tool for automating the development of management agents based on different frameworks e.g. OSI-SM and SNMP. The architectural decomposition of the generic agent is largely based on the OSIMIS GMS which also served as the basis for the relevant prototype.

[Katch95] describes work towards the enhancement of GDMO with behavioural specifications for automating the detection of pre-conditions that will result in the generation of a notification. GDMO augmentations have been proposed and the OSIMIS GDMO compiler and generic agent infrastructure have been modified to support those.

[Sidou95] describes an information model simulator that supports the testing of a newly developed GDMO model. This simulates the behaviour of managed objects and uses the OSIMIS agent infrastructure and support tools.

[Mazz96] describes extensions both to the agent and manager parts of OSIMIS that allow the realisation of interpreted managed and managing objects in the Scheme language. The goal is to achieve rapid prototyping of TMN applications by avoiding the compile and link cycle.

[Aner96] describes an architecture and experimental platform for the management of ATM Virtual Path Connections (VPCs). The relevant development was based on OSIMIS, while the author developed his own generic MIB browser using X-Windows Motif technology.

[Korm96] describes issues behind the design and implementation of the OSI Usage Metering SMF, using the relevant tools and facilities provided by OSIMIS. It also describes an associated case study for accounting CPU usage in a distributed workstation environment.

[Park96] describes the development of a Shared Management Knowledge (SMK) system using two different approaches, one based on OSIMIS and one based on CORBA. It then presents a comparative performance evaluation of the two approaches, concluding that the CORBA implementation is relatively faster.

[Nataf97] investigates the issues behind GRM-based relationship representation [X725] and proposes a prototype as a first step towards a complete support tool for relationship management. The relevant prototype is based on OSIMIS.

[Ramo97] proposes a novel approach towards providing access control services [X741] in OSI-SM/TMN environments by exploiting the use of allomorphism. The relevant experimentation was based on OSIMIS.

[Seitz97] investigates the use of a meta managed object specification language that can be mapped onto specific languages and access protocols. The OSIMIS uxObj example class is used to demonstrate the features of the meta-language while the OSIMIS environment has been used for the relevant prototype.

It should be finally added that OSIMIS has been used by more than one hundred companies and research institutions around the world, whose research and development work may have not resulted in publications.

Appendix B: The Object Modelling Technique Notation

The Object Modelling Technique (OMT) notation [Rumb91] has been used throughout this thesis for describing object class relationships. Figure B-1 depicts the OMT notation for the relationships used in this thesis.

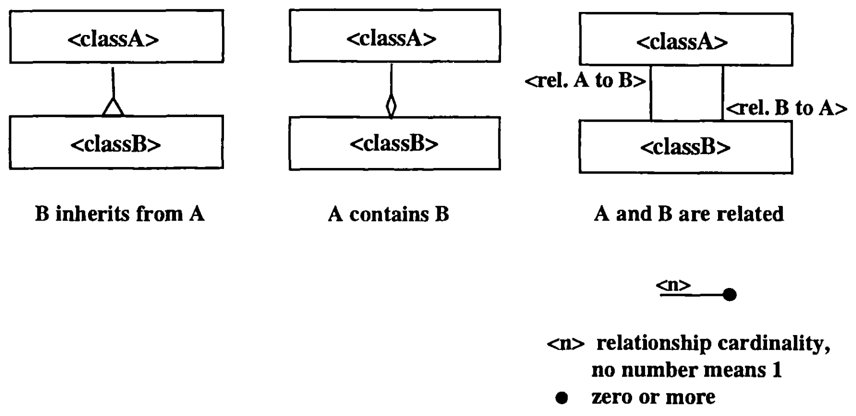


Figure B-1 OMT Notation for Inheritance, Containment and Other Relationships

Appendix C: Specification of the Managed Object Classes Used in the Examples

In Chapters 3 and 4, the *uxObj* and *simpleStats* object classes were used in the examples to demonstrate aspects the proposed infrastructure. Their complete specification is included in this appendix, first in OSI-SM GDMO/ASN.1and then in CORBA IDL. The IDL specifications have been derived from the GDMO/ASN.1 ones, following the JIDM guidelines [JIDM95]. It should be noted an OSI-SM implementation of those classes is part of the OSIMIS-4.0 distribution [Pav95b].

C.1 Specification in GDMO/ASN.1

In GDMO/ASN.1 specifications, a set of GDMO templates [X722] specify first the object-oriented aspects of the managed object class. An ASN.1 [X208] module follows, specifying the types of the attributes, actions, notifications and the object identifiers used when registering the various entities in the GDMO specification.

The root of the GDMO inheritance hierarchy is the *top* class [X721], which is not included in this specification. Both the uxObj and simpleStats classes are positioned in the MIT under instances of the *system* class [X721], which is the MIT root. The system class is not included in this specification.

Note also that the *gauge* attribute and the *objectCreation*, *objectDeletion* and *attributeValueChange* notifications and their syntaxes are imported from the Definition of Management Information recommendation [X721], so they are not included in the specification.

```
-- THE uxObj CLASS

-- Class and Package Templates

uxObj MANAGED OBJECT CLASS
DERIVED FROM        top;
CHARACTERIZED BY    uxObjPackage;
REGISTERED AS       { uclManagedObjectClass 50 };

uxObjPackage PACKAGE
ATTRIBUTES
    uxObjId                 GET,
    sysTime                 GET,
    wiseSaying              GET-REPLACE REPLACE-WITH-DEFAULT
                            DEFAULT VALUE
                            UCL-ASN1Module.defaultUxObjWiseSaying,
    nUsers                  GET;
ACTIONS
    echo;
NOTIFICATIONS
    objectCreation,
    objectDeletion,
    attributeValueChange;    -- when setting the wiseSaying attribute
REGISTERED AS       { uclPackage 50 };


-- Name Binding Template, positions uxObj instances in the MIT

uxObj-system NAME BINDING
SUBORDINATE OBJECT CLASS    uxObj AND SUBCLASSES;
NAMED BY
SUPERIOR OBJECT CLASS       system AND SUBCLASSES;
WITH ATTRIBUTE              uxObjId;
CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
DELETE                      ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS               { uclNameBinding 50 };


-- Attribute Templates

uxObjId ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.SimpleNameType;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 501 };

sysTime ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.Time;
MATCHES FOR EQUALITY, ORDERING;
REGISTERED AS       { uclAttributeID 502 };

wiseSaying ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.String;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 503 };

nUsers ATTRIBUTE
DERIVED FROM X721:gauge;
REGISTERED AS       { uclAttributeID 504 };


-- Action Templates

echo ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.String;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.String;
REGISTERED AS       { uclAction 501 };
```

```
-- THE simpleStats CLASS


-- Class and Package Templates

simpleStats MANAGED OBJECT CLASS
DERIVED FROM        top;
CHARACTERIZED BY    simpleStatsPackage;
REGISTERED AS       { uclManagedObjectClass 70 };

simpleStatsPackage PACKAGE
ATTRIBUTES
    simpleStatsId   GET;
ACTIONS
    calcSqrt,
    calcMeanStdDev;
REGISTERED AS       { uclPackage 70 };


-- Name Binding Template, positions uxObj instances in the MIT

simpleStats-system NAME BINDING
SUBORDINATE OBJECT CLASS    simpleStats AND SUBCLASSES;
NAMED BY
SUPERIOR OBJECT CLASS       system AND SUBCLASSES;
WITH ATTRIBUTE              simpleStatsId;
CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
DELETE;
REGISTERED AS               { uclNameBinding 70 };


-- Attribute Templates

simpleStatsId ATTRIBUTE
WITH ATTRIBUTE SYNTAX
    UCLAttribute-ASN1Module.SimpleNameType;
MATCHES FOR EQUALITY, SUBSTRINGS, ORDERING;
REGISTERED AS       { uclAttributeID 701 };


-- Action Templates

calcSqrt ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.Real;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.Real;
REGISTERED AS       { uclAction 701 };

calcMeanStdDev ACTION
MODE CONFIRMED;
WITH INFORMATION SYNTAX
    UCLAttribute-ASN1Module.RealList;
WITH REPLY SYNTAX
    UCLAttribute-ASN1Module.MeanStdDev;
REGISTERED AS       { uclAction 701 };
```

```
-- THE ASN.1 MODULE

UCL-ASN1Module DEFINITIONS ::=

BEGIN


uclManagedObjectClass OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 3}
uclNameBinding OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 6}
uclPackage OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 4}
uclAttributeID OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 7}
uclAction OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 9}
uclNotification OBJECT IDENTIFIER ::=
    {joint-iso-itu(2) mgmt(37) ucl(1) uclsmi(1) 10}


-- SimpleNameType could have been imported from the X.721 ASN.1 module
-- but is included here to demonstrate use of the CHOICE ASN.1 type and
-- its translation to CORBA IDL in the next section

SimpleNameType ::= CHOICE
{
    num     INTEGER,
    str     GraphicString
}


-- GraphicString and UTCTime are base ASN.1 types

String ::= GraphicString

Time ::= UTCTime


-- REAL is a base ASN.1 type

RealList ::= SET OF REAL

MeanStdDev ::= SEQUENCE
{
    mean    REAL,
    stdDev  REAL
}

-- default values

defaultUxObjWiseSaying String ::= "Hello World"
END
```

C.2 Specification in CORBA IDL

The same specifications are included here in CORBA IDL, based on the JIDM translation rules. The translation follows the spirit rather than the letter of the rules. For example, no exceptions have been associated with attribute access methods (they are unnecessary).

The nUsers attribute is of gauge type which has associated ObservedValue ASN.1 syntax [X721]. This maps to the ObservedType_t IDL type which is not explicitly included in this specification. Note also that the notification interface for the uxObj class is not included.

The *i_uxObj* and *i_simpleStats* interfaces inherit from *i_top* one, which results from translating the GDMO *top* class [X721]. The i_top interface has been depicted in the Code 4-1 caption in Chapter 4 and inherits subsequently from the*i_ManagedObject* interface.

```
// THE i_uxObj INTERFACE

// the argument types for operations are required first in IDL,
// in a similar fashion to programming languages

// SimpleNameType: integer or string

enum SimpleNameTypeChoice {
    numberChoice,
    stringChoice
};

union SimpleNameType_t switch (SimpleNameTypeChoice) {
    case numberChoice: long   num;
    case stringChoice: string str;
};

typedef string UTCTime_t; // string with well-defined format

// note there is no name binding in CORBA

// the invalidArgumentValue exception is part of the
// _ManagedObject interface and is defined as follows:
//
// exception invalidArgumentValue {
//     any argumentValue
// };


interface i_uxObj  : i_top
{
    SimpleNameType_t      uxObjId_get ();
    UTCTime_t             sysTime_get ();
    string                wiseSaying_get ();
    void                  wiseSaying_set (in string);
    string                wiseSaying_setDefault ();
    ObservedValue_t       nUsers_get ();
    void                  echo (in string, out string)
                              raises (invalidArgumentValue);
                              // never raised
};


// the notifications are part of a separate interface
// THE i_simpleStats INTERFACE

// SimpleNameType_t has been previously defined

typedef sequence<real> RealList_t;

typedef struct MeanStdDev_t
{
    real     mean;
    real     stdDev;
}

interface i_simpleStats : i_top
{
    // invalidArgumentValue is a standard CMIS action error which
    // is mapped to an IDL exception - it should be normally defined
    // as part of the i_ManagedObject interface

    SimpleNameType_t      simpleStatsId_get ();
    void                  calcSqrt (
                                in real, out real
                          ) raises (invalidArgumentValue);
    void                  calcMeanStdDev (
                                in RealList_t,
                                out MeanStdDev_t
                          ) raises (invalidArgumentValue);
                          // never raised
};
```

According to the JIDM rules, every action method with an *in* argument may raise an invalidArgumentValue exception. While this is the case for the calcSqrt method, e.g. for a negative number, this exception is never raised for the calcMeanStdDev method. The same is true for the i_uxObj echo method. In short, automatic translation cannot exploit semantics associated with the GDMO object class.

Appendix D: A String Language for CMIS Filters

A CMIS filter [X711] is a recursive ASN.1 type that supports tree-like expressions, with attribute value assertions in the leaves associated with boolean operators. The precise specification in ASN.1 is the following:

```
CMISFilter ::= CHOICE {
    item      [8]  EXPLICIT FilterItem,
    and       [9]  IMPLICIT SET OF CMISFilter,
    or        [10] IMPLICIT SET OF CMISFilter,
    not       [11] EXPLICIT CMISFilter
}

FilterItem ::= CHOICE {
    equality        [0] IMPLICIT Attribute
    substrings      [1] IMPLICT SEQUENCE OF CHOICE
      initialString       [0] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
      anyString           [1] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
      finalString         [2] IMPLICIT SEQUENCE {
                              attributeId AttributeId,
                              string ANY DEFINED BY AttributeId
                              },
    greaterOrEqual  [2] IMPLICIT Attribute,
    lessOrEqual     [3] IMPLICIT Attribute,
    present         [4] AttributeId,
    subsetOf        [5] IMPLICIT Attribute,
    supersetOf      [6] IMPLICIT Attribute,
    nonNullSetIntersection
                    [7] IMPLICIT Attribute
}
```

In the above specification, an attribute comprises an attribute ID and attribute value as defined in [X711]. When mapping this type to a C++ class through the O-O ASN.1 compiler, the user has to “fill it in” with the attributes, values and boolean operators. This is a both tedious and error-prone procedure, especially for complex filter expressions. A string-based user-friendly “language” can be used, which can be parsed to create the associated C++ object instance. The OSIMIS string-based language for CMIS filters is described semi-formally below.

A filter expression is always (<cmisFilter>) where <cmisFilter> is one of: <andFilter>, <orFilter>, <notFilter> or <filterItem>. The characters used to represent the logical operators are: & for AND, | for OR and ! for NOT. The <andFilter>, <orFilter>, <notFilter> and <filterItem> are as follows:

- < andFilter> has the form: ((<cmisFilter>) & (<cmisFilter>) ...);
- < orFilter> has the form: ((<cmisFilter>) | (<cmisFilter>) ...);
- <notFilter> has the form: (!(<cmisFilter>); and
- <filterItem> has one of two forms:
    1. <attributeName>) for creating a filter item with the "present" assertion; and
    2. (<attributeName><assertionType><attributeValue>) for all the other assertion types.

Table D-1 shows the lexemes used for the filter item assertions. With the substrings operator, the * character can be used as a wild card, in a similar fashion to the UNIX shell. The attribute values should follow the string representations according to the print method for that ASN.1 type.

Lexeme	**Assertion Type**
=	equality
:=	substrings (* is the wild card)
>=	greater or equal
<=	less of equal
:<	subset of
:>	superset of
><	non-null set intersection

Table D-1 Lexemes Used for CMIS Filter Assertions

Examples of filter expressions are:

```
((objectClass=log) & (administrativeState=unlocked))
((objectClass=routeEntry) & (nextHopAddr=X)
((wiseSaying=*ello*) | (!(wiseSaying=*ello), (nUsers >= 10)
```

In this way, sophisticated filter expressions can be easily constructed. This string notation was devised by the author together with S. Bhatti of UCL who implemented the parser.

Appendix E: Lightweight CMIS/P Specification

The concepts behind the Lightweight CMIP (LCMIP) were described in section 3.3.2.4 of Chapter 3. The specification of the GetArgument and GetResult LCMIP types is include below in order to demonstrate the simplifications compared to CMIP [X711]. These simplifications are briefly discussed after the specification.

```
GetArgument ::= SEQUENCE {
    baseManagedObjectClass      ObjectClass,
    baseManagedObjectInstance   ObjectInstance,
    accessControl               AccessControl,
    synchronization             CMISSync,
    scope                       Scope,
    filter                      CMISFilter,
    attributeIdList             AttributeIdList
}

GetResult ::= SEQUENCE {
    managedObjectClass          ObjectClass,
    managedObjectInstance       ObjectInstance,
    currentTime                 GeneralizedTime,
    getAttributeList            SET OF GetAttribute
}

GetAttribute ::= SEQUENCE {
    attributeId    AttributeId,
    attributeValue AttributeValue,
    errorStatus    ENUMERATED {
                          noError (0),
                          accessDenied (2),
                          noSuchAttribute (5)
                   }
}

CMISSync ::= ENUMERATED {       -- same as in [X711]
    bestEffort (0),
    atomic (1)
}

Scope ::= SEQUENCE {            -- simplified version of [X711]
    type     ENUMERATED {
                 baseObject (0),
                 firstLevelOnly (1),
                 wholeSubtree (2),
                 individualLevel (3),
                 baseToNthLevel (4)
             },
    level    INTEGER
}

CMISFilter ::= GraphicString    -- the OSIMIS string notation

ObjectClass ::= GraphicString   -- textual name
ObjectInstance ::= LCMIPDN      -- global / local distinction
                                -- through agreed prefixes
LCMIPDN ::= GraphicString       -- OSIMIS/ISODE convention
                                -- e.g. logId=1@logRecordId=5
AccessControl ::= OCTET STRING
GeneralizedTime ::= GraphicString -- with internal structure
AttributeId ::= GraphicString   -- textual name
AttributeValue ::= GraphicString -- the pretty-printed string

GraphicString ::= OCTET STRING  -- for easy encoding/decoding
```

As it can be seen from the specification, LCMIP introduces a number of important simplifications. The ObjectClass and AttributeId types are string names instead of OIDs. The ObjectInstance is a string with internal structure in the form used in OSIMIS/ISODE. The same is the case with the CMISFilter which is a string according to the conventions described in Appendix D. The Scope is a simplified version of the CMIP Scope which avoids the ASN.1 CHOICE type. The AttributeValue is a well-agreed pretty-printed string representation for a particular type. Finally, the GraphicString type is encoded as ASN.1 OCTET STRING for the ease of encoding / decoding.

In summary, LCMIP is a lightweight version of CMIP that uses strings as much as possible and avoids tags, optional elements and other ASN.1 aspects which complicate encoding and decoding. It should be possible to implement LCMIP by hand, without the need for ASN.1 compilers which inevitably introduce inefficiencies.

Acronyms and Abbreviations

AAA	Autonomous Administrative Authority
AAP	Autonomous Administrative Point
ACSAP	Association Control Service Access Point
ACSE	Association Control Service Element
ACTS	Advanced Communications Technologies and Services
ADF	Access Decision Function
AET	Application Entity Title
AH	Agent Handle
AI	Artificial Intelligence
AIP	Advanced Information Processing
ANSA	Advanced Networked Systems Architecture
API	Application Programming Interface
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
AVA	Attribute Value Assertion
BAF	Broadband Access Facilities (RACE Project No. 2024)
BEO	Basic Engineering Object
BER	Basic Encoding Rules
B-ISDN	Broadband ISDN
BM	Business Management
BML	Business Management Layer
CDS	Cell Directory Service
CEU	Commission of the European Union
CIOP	Common Inter-Operability Protocol
CLNP	Connection-Less Network Protocol
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
CMOL	CMIP Over LLC

CMOT	CMIP Over TCP
CORBA	Common Object Request Broker Architecture
COSS	Common Object Service Specification
COTP	Connection Oriented Transport Protocol
COTS	Connection Oriented Transport Service
DAF	Directory Access Function
DAP	Directory Access Protocol
DAS	Directory Access Service
DASE	Directory Access Service Element
DCE	Distributed Computing Environment
DCN	Data Communication Network
DCOM	Distributed Component Object Model
DIB	Directory Information Base
DII	Dynamic Invocation Interface
DIT	Directory Information Tree
DME	Distributed Management Environment
DMI	Definition of Management Information
DO	Directory Object
DPE	Distributed Processing Environment
DPL	Distributed Processing Language
DSA	Directory Service Agent
DSF	Directory System Function
DSOM	Distributed Systems: Operations and Management
DSP	Directory System Protocol
DTP	Distributed Transaction Processing
DTS	Distributed Time Service
DUA	Directory User Agent
EFD	Event Forwarding Discriminator
EM	Element Management
EML	Element Management Layer
EP	Event Processing
ER	Encoding Rules
ESPRIT	European Specific Research in Information Technology
ETSI	European Telecommunications Standards Institute

EURESOM	European Institute for Research and Strategic Studies in Telecommunications
FCAPS	Fault Configuration Accounting Performance Security
FTAM	File Transfer Access Method
FTSE	File Transfer Service Element
FU	Functional Unit
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Inter-Operability Protocol
GMS	Generic Managed System
GOM	Generic Object Model
GRM	General Relationship Model
GSM	Global System for Mobile Telecommunications
GUI	Graphical User Interface
GULS	Generic Upper Layer Security
HTTP	HyperText Transfer Protocol
IBC	Integrated Broadband Communications
ICF	Information Conversion Function
ICM	Integrated Communications Management (RACE Project No. 2059)
IDL	Interface Definition Language
IDMIS	Inter-Domain Management Information Service
IDSM	Integrated Distributed Systems Management (ESPRIT Project No. 6311)
IETF	Internet Engineering Task Force
IIMC	ISO/ITU-T Internet Management Coexistence
IM	Integrated Management
IMF	Intelligent Monitoring Function
IN	Intelligent Network
INCA	Integrated Network Communicaiions Architecture (ESPRIT Project No. ____)
IOR	Inter-Operable Reference
IP	Internet Protocol
IPC	Inter-Process Communication
IQA	Internet Q-Adapter
ISDN	Integrated Services Digital Network
IS&N	Intelligence in Services and Networks
ISO	International Standards Organisation
ISODE	ISO Development Environment

ISP	Internet Service Provider
ITU-T	International Telecommunications Union
JIDM	NMF - X/Open Joint Inter-Domain Management task force
JNSM	Journal of Network and Systems Management
KS	Knowledge Source
LAN	Local Area Network
LCMIP	Lightweight CMIP
LDAP	Lightweight DAP
LDN	Local Distinguished Name
LLA	Logical Layered Architecture
LLC	Logical Link Control
LPP	Lightweight Presentation Protocol
MAF	Management Application Function
MASIF	Mobile Agent System Interoperability Facility
MB	Management Broker
MCF	Message Communication Function
MD	Mediation Device
MF	Mediation Function
MIB	Management Information Base
MIDAS	Management in a Distributed Application and Service Environment (ESPRIT Project No. ____)
MIM	Management Information Model
MISA	Management of IntegratedSDH and ATM (ACTS Project No. 80)
MIT	Management Information Tree
MO	Managed Object
MOC	Managed Object Class
MOH	Managed Object Handle
MSAP	Management Service Access Point
MUIB	Management Unit Information Base
NE	Network Element
NEF	Network Element Function
NEMESYS	Network Management Using Expert Systems (RACE Project No. 1005)
NM	Network Management
NML	Network Management Layer

NMF	Network Management Forum
NP	Notification Processing
NRIM	Network Resource Information Model
NS	Name Server
ODMA	Object Distributed Management Architecture
ODP	Open Distributed Processing
OID	Object Identifier
OIM	OSI Internet MIB
OMA	Object Management Architecture
OMG	Object Management Group
OMNIPoint	Open Management Interoperability Point
OMT	Object Modelling Technique
OOD	Object-Oriented Design
OOI	Object-Oriented Interface
ORB	Object Request Broker
OS	Operations System
OSF	Operations System Functbn
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSI-RM	OSI Reference Model
OSI-SM	OSI Systems Management
OSIMIS	OSI Management Information Service
PC	Personal Computer
PDU	Protocol Data Unit
PE	Presentation Element
PMO	Proxy Managed Object
PREPARE	Pre-Pilot in Advanced Resource Management (RACE Project No. 2004)
PRISM	Pan-European Reference Configuration for IBC Service Management (RACE Project No. 2041)
PROOF	Primary Rate ISDN OSI Office Facilities (ESPRIT Project No.____)
PROSPECT	It is not an acronym (ACTS Project No. 52)
PSAP	Presentation Service Access Point
QA	Q-Adapter
QAF	Q-Adapter Function

QoS	Quality of Service
RA	Resource Adapter
RACE	Research in Advanced Communications in Europe
RAM	Read Access Memory
REFORM	REsource and Fault restORation and Management in ATM (ACTS Project No. 10135)
RDN	Relative Distinguished Name
RFC	Request For Comments
RMI	Remote Method Invocation
RMIB	Remote Management Information Base
ROSAP	Remote Operations Serviœ Access Point
ROSE	Remote Operations Service Element
RPC	Remote Procedure Call
SAP	Service Access Point
SDH	Synchronous Digital Hierarchy
SDL	Specification and Description Language
SM	Service Management
SMA	Systems Management Agent
SMAE	Systems Management Application Entity
SMAP	Systems Management Application Process
SMAS	Systems Management Application Service
SMASE	Systems Management Application Service Element
SMF	Systems Management Function
SMI	Structure of Management Informatbn
SMIB	Shadow Management Information Base
SMK	Shared Management Knowledge
SML	Service Management Layer
SMO	Shadow Managed Object
SNMP	Simple Network management Protocol
SONET	Synchronous Optical Network
SS7	Signalling System number 7
SSAP	Session Service Access Point
SSMIP	Simple String-based Management Information Protocol
TBOS	Telemetry Byte-Oriented Serial protocol

TCP	Transmission Control Protocol
TINA	Telecommunications Information Networking Architecture
TL-1	Transaction Language 1
TMN	Telecommunications Management Network
TOMQAT	Total Quality of Service Management in ATM (RACE Project No. 2116)
TP	Transaction Protocol
TSAP	Transport Service Access Point
UCL	University College London
UCL CS	UCL Computer Science
UISF	User Interface Support Function
UTC	Universal Time
VITAL	Validation of Integrated Telecommunications Architecture for the Long-term (ACTS Project No. 187)
VCC	ATM Virtual Channel Connection
VPC	ATM Virtual Path Connection
VPCRM	Virtual Path Connection and Routing Management
VPN	Virtual Private Network
WS	Workstation
WSF	Workstation Function
WSSF	Workstation Support Function
WWW	World Wide Web
XDR	eXternal Data Representation
XMP	X/Open Management Protocols
XOM	X/Open OSI-Abstract-Data Manipulation

List of Figures

List of Tables

List of Code Captions

Bibliography

[Aho83] A. Aho, J. Hopcroft, J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983

[Aida94] S. Aidarous, T. Plevyak, ed., *Telecommunications Network Management into the 21st Century: Techniques, Standards, Technologies and Applications*, IEEE Press, 1994

[Aida97] S. Aidarous, T. Plevyak, ed., *Telecommunications Network Management: Technologies and Implementations,* IEEE Press, 1997

[Aner96] N. Anerousis, A. Lazar, *An Architecture for Managing Virtual Circuit and Virtual Path Services in ATM Networks*, Journal of Network and Systems Management (JNSM), Vol. 4, No. 4, pp. 425-455, Plenum Publishing, 1996 -
early version in [IM-IV], pp. 370-384, Chapman & Hall, 1995

[ANSA89a] *The ANSA Reference Manual*, APM Ltd., 1989

[ANSA89b] *ANSA: An Engineer's Introduction to the Architecture*, APM Ltd., 1989

[Autr94] M. Autrata, C. Strutt, *Distributed Management Environment (DME) Framework and Design*, in [Slom94], pp. 605-627, Addison-Wesley, 1994

[BenN94] R. Ben-Natan, *CORBA: A Guide to the Common Object Request Broker Architecture*, McGraw-Hill, 1995

[Bern93] L. Bernstein, C. Yuhas, *Managing Telecommunications Networks*, IEEE Network, November 1993

[Besa89] L. Besaw, U. Warrier, *Common Management Information Service and Protocol over TCP/IP (CMOT)*, IETF RFC 1095, 1989

[Bhat95] S.N. Bhatti, G. Knight, D. Gurle, P. Rodier, *Secure Remote Management*, in [IM-IV], pp. 156-169, Chapman & Hall, 1995

[Bhat96] S.N. Bhatti, K. McCarthy, G. Knight, G.Pavlou, *Secure Management Information Exchange*, Journal of Network and System Management, Vol. 4, No. 3, pp. 251-257, Plenum Publishing, 1996

[Birr84] A. Birrell, B. Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, vol. 2, pp. 39-59, February 1984

[BISDN] ITU-T Rec. I.320-321, *Broadband ISDN Reference Model*

[I320] *ISDN Protocol Reference Model*, 1991

[I321] *B-ISDN Protocol Reference Model and its Application*, 1991

[Bjer94] L.H. Bjerring, M. Tschichholz, *Requirements of Inter-Domain Management and their Implications for TMN Architecture*, in [ISN94], pp. 193-206, Springer, 1994

[Black92] U. Black, *Network Management Standards - The OSI, SNMP and CMOL Protocols*, McGraw-Hill, 1992

[Booch91] G. Booch, *Object-Oriented Design With Applications*, Benjamin Cummings, 1991

[Chat97] T. Chatt, M. Curry, U. Holberg, J. Seppa, *TMN/C++: an Object-Oriented API for GDMO, CMIS and ASN.1*, in [IM-V], pp. 177-191, Chapman & Hall, 1997

[Clark85] D. Clark, *The Structuring of Systems Using Upcalls*, ACM Operating Systems Review, pp. 171-180, December 1985

[Cohen94] R. Cohen, *The Telecommunications Management Network*, in [Slom94], pp. 217-243, Addison-Wesley, 1994

[Coll89] W. Collins, *OSI Management Service Elements, Protocols and Application Layer Structure*, in [IM-I], pp. 119-131, Elsevier, 1989

[CORBA] Object Management Group, *The Common Object Request Broker: Architecture and Specification (CORBA)*, Version 2.0, 1995

[COSS] Object Management Group, *CORBA Services: Common Object Services Specification (COSS)*, Revised Edition, 1995

[Coul88] G. Coulouris, J. Dollimore, *Distributed Systems - Concepts and Design*, Addison-Wesley, 1988

[Cox86] B. Cox, A. Novobilski, *Object-Oriented Programming: an Evolutionary Approach*, Addison-Wesley, 1986

[Crist95] H. Christensen, E. Colban, *Information Modelling Concepts*, TINA-C baseline document TB_EAC.001_1.2_94, 1995

[Crow93] J. Crowcroft, *Remote Procedure Call: Not a Panacea for Internet-wide Distributed Computing Problems*, UCL CS Research Note RN/93/12, 1993

[DCE] Open Software Foundation, *An Introduction to the OSF Distributed Computing Environment (DCE)*, 1992

[DCOM] Microsoft, *The Component Object Model Specification*, http://www.microsoft.com/oledev/olecom/title.htm

[Dens91] M. Densmore, *Providing CMIS Services in DECmcc*, in [IM-II], pp. 313-326, Elsevier, 1991

[Deri95] L. Deri, E. Mattei, *An Object-Oriented Approach to the Implementation of OSI Management*, Computer Networks and ISDN Systems, Vol. 27, pp. 1367-1385, 1995

[Deri97] L. Deri, *A Component-based Architecture for Open, Independently Extensible Distributed Systems*, PhD Thesis, Bern University, Switzerland, July 1997

[Ditt95] A. Dittrich, *The ANDROMEDA Platform: an Object-Oriented Development and Run-time Environment for Management Services*, in [DSOM95], pp. 2.3/1-9, 1995

[DME] Open Software Foundation, *The OSF Distributed Management Environment (DME) Architecture*, 1992

[Doss93] F. Dossogne, M.-P. Dupont, *A Software Architecture for Management Information Model Definition, Implementation and Validation*, in [IM-III], pp. 593-604. Elsevier, 1993

[DSOM94] Proc. of the 5th IFIP/IEEE International Workshop on *Distributed Systems: Operations and Management (DSOM'94)*, Toulouse, France, October 1994

[DSOM95] Proc. of the 6th IFIP/IEEE International Workshop on *Distributed Systems: Operations and Management (DSOM'95)*, Ottawa, Canada, October 1995

[DSOM96] Proc. of the 7th IFIP/IEEE International Workshop on *Distributed Systems: Operations and Management (DSOM'96)*, L'Aquila, Italy, October 1996

[DSOM97] A. Seneviratne, V. Varadarajan, P. Ray, ed., Proc. of the 8th IFIP/IEEE International Workshop on *Distributed Systems: Operations and Management (DSOM'97)*, ISBN 1 86365 3449, Sydney, Australia, October 1997

[DTP] ITU-T Rec. X.860, Information Technology - Open Systems Interconnection, *Distributed Transaction Processing Model,* 1992

[Ellis91] M. Ellis, B. Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, 1991

[Embry91] J. Embry, P. Manson, D. Milham, *Interoperable Network Management: OSI/NM Forum Architecture and Concepts*, in [IM-II], Elsevier, 1991

[Feri96] M. Feridun, L. Heusler, R. Nielsen, *Implementing OSI Agent/Managers for TMN*, IEEE Communications, Special Issue on Making TMN a Reality, vol. 34, no. 9, pp. 62-67, September 1996

[Fest93] O. Festor, G. Zorntlein, *Formal Description of Managed Object Behaviour - A Rule Based Approach*, in [IM-III], pp. 45-58, Elsevier, 1993

[Fest95] O. Festor, *MODE: a Development Environment for Managed Objects Based on Formal Methods*, in [IM-IV], pp. 616-628, Chapman & Hall, 1995

[Filip93] W. Filip, G. Kar, *NetView in a Heterogeneous Environment: Implementation of an Experimental Agent Structure*, in [IM-III], pp. 505-517, Elsevier, 1993

[Flau95] M. Flauw, P. Jardin, *Designing a Distributed Management Framework - an Implementer's Perspective*, in [IM-IV], pp. 506-519, Chapman & Hall, 1995

[FTAM] ISO 8571-1, *File Transfer Access and Management - Part 1: General Introduction*, 1988

[Fuen94] L.A. de la Fuente, J. Pavon, N. Singer, *Application of TINA-C Architecture to Management Services*, in [ISN94], pp. 273-284, Springer, 1994

[Garc96] E. Garcia-Lopez, *Distributed Management Facilities Architecture*, TINA-C baseline document TB_EGL.002_2.1_1996, 1996

[Geni95] G. Genilloud, D. Gay, *Accessing OSI Managed Objects from ANSAware*, in [DSOM95], pp. 1.2/1-9, 1995

[Geni96] G. Genilloud, *Towards a Distributed Architecture for Systems Management*, PhD Thesis no. 1588, Ecole Polytechnique Federale de Lausanne, Switzerland, December 1996

[Georg95] P. Georgatsos, D. Griffin, *Management Services for Performance Verification in Broadband Multi-service Networks*, in [ISN965], pp. 275-289, Springer, 1995

[Geri94] M. Gering, *Comparison of SNMP and CMIP Management Architectures*, in [Slom94], pp. 197-216, 1994

[GIOP] Object Management Group, *General Inter-Operability Protocol (GIOP)*, CORBA version 2.0, 1995

[Glith95] R. Glitho, S. Hayes, *Telecommunications Management Network: Vision vs. Reality*, IEEE Communications, pp. 47-52, March 1995

[Glith96] R. Glitho, S. Hayes, *Approaches for Introducing TMN in Legacy Networks: A Critical Look*, IEEE Communications, pp. 55-60, September 1996

[Gold83] A. Goldberg, D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, 1983

[GOM] ETSI Draft Rec., Telecommunications Management Network, *Generic Managed Object Class Library for the Network Level View (GOM)*, Version H2, 1995

[Grif95] D. Griffin, P. Georgatsos, *A TMN System for VPC and Routing Management in ATM Networks*, in [IM-IV], pp. 356-369, Chapman & Hall, 1995

[Grif96a] D. Griffin, G. Pavlou, *Integrated Communications Management Methodology for TMN System Design*, in [ICM], pp. 37-47, Crete University Press, 1996

[Grif96b] D. Griffin, P. Georgatsos, et al, *ATM Virtual Path Connection and Routing Management*, in [ICM], pp. 73-145, Crete University Press, 1996

[GULS] ITU-T Rec. X.830-833, Information Technology - Open Systems Interconnection, *Security: Generic Upper Layer Security (GULS)*, 1996

[G774] ITU-T Rec. G.774, *Synchronous Digital Hierarchy (SDH) - Management Information Model for the Network Element View*, 1992

[Hand95] T. Handegard, N. Mercouroff, *Computational Modelling Concepts*, TINA-C baseline document TP_HC.012_3.1_95, 1995

[Hars96] M. Harssema, *Integrating TMN and CORBA*, MSc Thesis, University of Twente, The Netherlands, October 1996

[Hauw97] L. Haw, Z. Canela, F. Voyer, *A CORBA-based TMN Prototype with Web Access*, in [DSOM97], pp. 81-93, ISBN 1 86365 3449, 1997

[Hier96] J. Hierro, J. Gonzalez, *Common Facilities for Systems Management*, Telefonica I+D Submission to the NMF - X/Open Joint Inter-Domain Management task force, 1996

[Holb95] U. Holberg, *High-Level Support for TMN Applications: the Object-Oriented Interface Generator and Run-time System - Overview*, IBM ENC, Heidelberg, 1995

[HTTP] T. Berners-Lee, R. Fielding, H. Nielsen, *Hypertext Transfer Protocol - HTTP/1.0*, IETF RFC 1945, 1996

[Huel97] J. Huelamo, H. Vanderstraeten, J.C. Garcia, G. Pavlou, *A TINA-based Prototype for Multimedia Multiparty Mobility Services*, in [ISN97], pp. 35-47, Springer, 1997

[Huit92] C. Huitema, G. Chave, *Measuring the Performance of an ASN.1 Compiler*, in Proc. IFIP TC6/WG6.5, Upper Layer Protocols, Architectures and Applications, ed. B. Plattner, pp. 99-112, Elsevier, 1992

[ICM] D. Griffin, ed., *Integrated Communications Management of Broadband Networks*, Crete University Press, 1996

[IIMC] NMF, *ISO/ITU-T and Internet Management Coexistence (IIMC)*

[NMF-026] *Translation of Internet MIBs to ISO/ITU-T GDMO MIBs*, 1994

[NMF-028] *ISO/ITU-T to Internet Management Proxy*, 1994

[IIOP] Object Management Group, *Internet Inter-Operability Protocol (IIOP)*, CORBA version 2.0, 1995

[IM-I] B. Meandzija, J. Westcott, ed., *Integrated Network Management I*, Proc. of the IFIP TC6/WG 6.6 Symposium on Integrated Network Management (ISINM'89), Boston, Elsevier, May 1989

[IM-II] I. Krishnan, W. Zimmer, ed., *Integrated Network Management II*, Proc. of the IFIP TC6/WG 6.6 and IEEE Communications Society Symposium on Integrated Network Management (ISINM'91), Washington D. C., Elsevier, April 1991

[IM-III] H.-G. Hegering, Y. Yemini, ed., *Integrated Network Management III*, Proc. of the IFIP TC6/WG 6.6 and IEEE Communications Society Symposium on Integrated Network Management (ISINM'93), San Francisco, Elsevier, April 1993

[IM-IV] A. Sethi, Y. Raynaud, F. Faure-Vincent, ed., *Integrated Network Management IV*, Proc. of the IFIP TC6/WG 6.6 and IEEE Communications Society Symposium on Integrated Network Management (ISINM'95), Santa Barbara, Chapman & Hall, April 1995

[IM-V] A. Lazar, R. Saracco, R. Stadler, ed., *Integrated Network Management V*, Proc. of the IFIP TC6/WG 6.6 and IEEE Communications Society Symposium on Integrated Network Management (IM'97), Santa Barbara, Chapman & Hall, May 1997

[ISN93] Proc. of the 1st International Conference on *Intelligence in Services and Networks (IS&N'93)*, Paris, France, November 1993

[ISN94] H.-J. Kugler, A. Mullery, N. Niebert, ed., *Towards a Pan-European Telecommunications Service Infrastructure*, Proc. of the 2nd International Conference on Intelligence in Services and Networks (IS&N'94), Aachen, Germany, Springer, September 1994

[ISN95] A. Clarke, M. Campolargo, N. Karatzas, ed., *Bringing Telecommunications Services to the People*, Proc. of the 3rd International Conference on Intelligence in Services and Networks (IS&N'95), Heraklion, Crete, Greece, Germany, Springer, October 1995

[ISN97] A. Mullery, M. Besson, M. Campolargo, R. Gobbi, R. Reed, ed., *Intelligence in Services and Networks: Technology for Cooperative Competition*, Proc. of the 4th International Conference on Intelligence in Services and Networks (IS&N'97), Cernobbio, Italy, Springer, May 1997

[ISODE] M. Rose, J. Onions, C. Robbins, *The ISO Development Environment: User's Manual, Version 7.0, Vol 1: Application Services, Vol 2: Underlying Services,* 1991

[I751] ITU-T Draft Rec. I.751, *Asynchronous Transfer Mode (ATM) - Management of the Network Element View*, 1995

[Jeff88] T. Jeffree, *A Review of OSI Management Standards*, Computer Networks and ISDN Systems, No. 16, Elsevier, 1988

[Jeff92] T. Jeffree, A. Langsford, C. Sluman, J. Tucker, J. Westgate, *Technical Guide for OSI Management*, NCC Blackwell, 1992

[Jeff94] T. Jeffree, *Guidelines for the Definition of Managed Objects*, in [Slom94], pp. 131-164, Addison0Wesley, 1994

[JIDM95] NMF - X/Open, Joint Inter-Domain Management (JIDM) Specifications, *Specification Translation of SNMP SMI to CORBA IDL, GDMO/ASN.1 to CORBA IDL and IDL to GDMO/ASN.1*, 1995

[John94] P. Johnson, *Domestic and International Open Systems Interconnection Management Standards*, in [Aida94], pp. 122-135, IEEE Press, 1994

[Jord93] D. Jordaan, M. Paterok, *Event Correlation in Heterogeneous Networks Using the OSI Management Framework*, in [IM-III], pp. 683-695, Elsevier, 1993

[Katch95] M. Katchabaw, M. Bauer, J. Hong, *Behavioural Specification and Notification Enhancements to GDMO*, in [DSOM95], pp 6.3/1-10, 1995

[Kern78] B. Kernighan, D. Richie, *The C Programming Language*, Prentice-Hall, 1978

[Kern84] B. Kernighan, R. Pike, *The UNIX Programming Environment*, Prentice-Hall, 1984

[Kill88] S. Kille, *The QUIPU Directory Service*, in Proc. of the 4th International Symposium on Computer Message Systems, ed. E. Stefferud, pp. 173-185, Costa Mesa, September 1988

[Kill91] S. Kille, *Implementing X.400 and X.500: the PP and QUIPU Systems*, Artech House, 1991

[Kits95] B. Kitson, *CORBA and TINA: the Architectural Relationships*, in [TINA95], pp. 371-386, 1995

[Kler88] M. Klerer, *The OSI Management Architecture: an Overview*, IEEE Network, Vol. 2, No. 2, March 1988

[Kler93] M. Klerer, *System Management Information Modelling*, IEEE Communications, pp. 38-44, May 1993

[Knig89] G. Knight, *The INCA Network Management System*, Connexions - The Interoperability Report, Vol. 3, No. 3, pp. 27-32, 1989

[Knig90] G. Knight, *The Open Management of Limited Systems*, Proc. of the International Network Management Conference, pp. 51-60, Blenheim-Online, Birmingham, UK, 1990

[Korm96] L. Korman, C. Westphall, A. Coser, *OSI Usage Metering Function in the OSIMIS Management Platform*, Journal of Network and Systems Management (JNSM), Vol. 4, No. 3, Plenum Publishing, 1996
also early version in [DSOM95], pp. 2.3/1-11, 1995

[Kram94] J. Kramer, *Distributed Systems*, in [Slom94], pp. 47-66, Addison-Wesley, 1994

[Laba91a] L. LaBarre, *OSI Event Generation, Reporting and Logging*, in [IM-II], pp. 227-242, Elsevier, 1991

[Laba91b] L. LaBarre, *OSI Internet Management: Management Information Base*, IETF RFC 1214, 1991

[Lang94] A. Langsford, *OSI Management Model and Standards*, in [Slom94], pp. 69-93, Addison-Wesley, 1994

[LDAP] W. Yeong, T. Howes, S. Kille, *Lightweight Directory Access Protocol (LDAP)*, IETF RFC 1777, 1995

[Lewis95] D. Lewis, S. O'Connell, W. Donnelly, L. Bjerring, *Experiences in Multi-Domain Management System Development*, in [IM-IV], pp. 494-505, Chapman & Hall, 1995

[Lioup94] Z. Lioupas, Y. Manolessos, M. Thelogou, *Structuring Principles for Total Quality of Service Management in IBCN*, in [ISN94], pp. 445-454, Springer, 1994

[Marc95] J.S. Marcus, *Icaros, Alice and the OSF DME*, in [IM-IV], pp. 83-92, Chapman & Hall, 1995

[Mazu96] S. Mazumdar, *Mapping of Common Management Information Services to OMG Common Object Services Specification*, ATT Bell Labs, TM # BL0112540-96.09.30-02, 1996

[Mazz96] S. Mazziotta, D. Sidou, *A Scheme-based Toolkit for the Fast Prototyping of TMN Systems*, in [DSOM96], pp. 8.3/1-9, 1996

[McCar95] K. McCarthy, G. Pavlou, S. Bhatti, J.N. DeSouza, *Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways*, in [IM-IV], pp. 440-453, Chapman & Hall, 1995

[Meyer88] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1988

[Milh89] D. Milham, K. Willets, *BT's Communications Management Architecture*, in [IM-I], pp. 109-116, Elsevier, 1989

[Moda90] A. Modaressi, A. Skoog, *Signalling System No. 7: a Tutorial*, IEEE Communications, pp. 19-35, July 1990

[Murr93] B. Murrill, *OMNIPoint: An Implementation Guide to Integrated Networked Information Systems Management*, in [IM-III], pp. 405-418, Elsevier, 1993

[M3010] ITU-T Rec. M.3010, *Principles for a Telecommunications Management Network (TMN),* Study Group IV,1996

[M3020] ITU-T Rec. M.3020, *TMN Interface Specification Methodology,* 1995

[M3100] ITU-T Rec. M.3100, *Generic Network Information Model,* 1995

[M3200] ITU-T Rec. M.3200, *TMN Management Services,* 1995

[M3300] ITU-T Rec. M.3300, *TMN Management Capabilities at the F Interface,* 1991

[M3400] ITU-T Rec. M.3400, *TMN Management Functions,* 1995

[Nakai91] S. Nakai, Y. Kiriha, Y. Ihara, S. Hasegawa, *A Development Environment for OSI Systems Management*, in [IM-II], pp. 157-168, Elsevier, 1991

[Nakak91] T. Nakakawaji, K. Katsuyama, N. Miyauchi, T. Mizuno, *MINT, an OSI Management Information Support Tool*, in [IM-II], pp. 845-855, Elsevier, 1991

[Nataf97] E. Nataf, O. Festor, L. Andrey, *RelMan: a GRM-based Relationship Manager*, in [IM-V], pp. 661-672, Chapman & Hall, 1997

[Natar95] N. Natarayan, *Network Resource Information Model (NRIM) Specification*, TINA-C baseline document TB_LR.010_2.1_95, 1995

[Neuf90] G. Neufeld, Y. Yang, *The Design and Implementation of an ASN.1-C Compiler*, IEEE Transactions on Software Engineering, Vol. 16, No. 10, 1990

[Newk94] O. Newkerk, M. Nihart, S. Wong, *The Common Agent - a Protocol Independent Management Agent*, IEEE Journal of Selected Areas in Communications (JSAC), Special Issue on Network Management, Vol. 11, No. 9, pp. 1346-1352, 1993

[Newn92] O. Newnan, *On Managing to be Managed*, in Proc. IFIP TC6/WG6.5, Upper Layer Protocols, Architectures and Applications, ed. B. Plattner, pp. 184-197, Elsevier, 1992

[NMF-025] NMF Forum 025, *The "Ensemble" Concepts and Format*, 1992

[ODP] ITU-T Draft Rec. X.901-904, Information Technology - Open Distributed Processing, *Basic Reference Model of Open Distributed Processing*

[X901] *Part 1: Overview*, 1995

[X902] *Part 2: Foundations*, 1995

[X903] *Part 3: Architecture*, 1995

[X904] *Part 4: Architectural Semantics*, 1995

[X9tr] *Draft ODP Trading Function*, 1994

[OmniPnt] Network Management Forum, *Discovering OMNIPoint - A Common Approach to the Integrated Management of Networked Information Systems*, Prentice Hall, 1993

[Oust94] J. Ousterhout, *The Tcl Language and the Tk Toolkit*, Addison-Wesley, 1994

[Park96] J.-T. Park, S.-H. Ha, J. Hong, *Implementation and Performance of a TMN Shared Management Knowledge System*, in [DSOM96], pp. 3.3/1-10, 1996
also to appear in the JNSM, Vol. 6 No. 2, 1998

[Pav91a] G. Pavlou, G. Knight, S. Walton, *Experience of Implementing OSI Management Facilities*, in [IM-II], pp. 259-270, Elsevier, 1991

[Pav91b] G. Pavlou, A. Mann, *Quality of Service Management in Integrated Broadband Communications: an OSI Management / TMN Based Prototype*, in Proc. of the 5th International TMN Conference, London, November 1991

[Pav92a] G. Pavlou, J. Cowan, J. Crowcroft, *A Generic Management Information Base Browser*, in Proc. IFIP TC6/WG6.5, Upper Layer Protocols, Architectures and Applications, ed. B. Plattner, pp. 221-232, Elsevier, 1992

[Pav92b] G. Pavlou, U. Harksen, A. Mann, *Experience of Modelling and Implementing TMN-based Quality of Service Management*, in The Management of Telecommunications Networks, ed. R. Smith, E.H. Mamdani, J. Callaghan, pp. 109-120, Ellis Horwood, 1992

[Pav93a] G. Pavlou, *Implementing OSI Management*, Tutorial presented in the 3rd IFIP/IEEE International Symposium on Integrated Network Management (ISINM'93), Proc. in [IM-III], San Francisco, April 1993, ftp://cs.ucl.ac.uk/osimis/tutorial-isinm93.ps.Z

[Pav93b] G. Pavlou, S. Bhatti, G. Knight, *The OSI Management Information Service User's Manual Version 1.0 (for OSIMIS version 3.0)*, University College London, January 1993, ftp://cs.ucl.ac.uk/osimis-manual-1.ps.Z

[Pav93c] G. Pavlou, S. Bhatti, G. Knight, *Automating the OSI to Internet Management Conversion through the Use of an Object-Oriented Platform*, in Proc. IFIP TC6/WG6.4, Advanced Information Processing Techniques for LAN and MAN Management, pp. 245-260, Elsevier, 1993

[Pav94a] G. Pavlou, *The Telecommunications Management Network (TMN)*, Tutorial presented in the 2nd International Conference on Intelligence in Services and Networks (IS&N'94), Proc. in [ISN94], Aachen, Germany, September 1994

[Pav94b] G. Pavlou, T. Tin, A. Carr, *High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding*, in [ISN94], pp. 219-230, Springer, 1994

[Pav95a] G. Pavlou, G. Knight, K. McCarthy, S. Bhatti, *The OSIMIS Platform: Making OSI Management Simple*, in [IM-IV], pp. 480-493, Chapman & Hall, 1995 -
also early version *The OSIMIS TMN Platform: Support for Multiple Technology Integrated Management Systems*, in [ISN93], 1993

[Pav95b] G. Pavlou, G. Knight, T.Tin, K. McCarthy, J. Cowan, S. Bhatti, *The OSI Management Information Service User's Manual Version 4.0 (for OSIMIS version 4.0)*, University College London, March 1995

[Pav95c] G. Pavlou, *LCMIP: a Lightweight Protocol Architecture and Specification for Data and Telecommunications Network Management*, UCL CS Research Note RN/95/61, 1995

[Pav96a] G. Pavlou, D. Griffin, *TMN Architecture Issues*, in [ICM], pp. 49-71, Crete University Press, 1996

[Pav96b] G. Pavlou, et al., *The OSIMIS TMN Platform*, in [ICM], pp. 267-307, Crete University Press, 1996

[Pav96c] G. Pavlou, G. Mykoniatis, J. Sanchez, *Distributed Intelligent Monitoring and Reporting Facilities*, IEE Distributed Systems Engineering Journal (DSEJ), Special Issue on Management, vol. 3, no. 2, pp. 124-135, IOP Publishing, 1996 -
also an early version in [ISN95], pp. 430-444, Springer, 1995

[Pav96d] G. Pavlou, T. Tin, *A CMIS-capable Scripting Language and Associated Lightweight Protocol for TMN Applications*, IEEE Communications, Special Issue on Making TMN a Reality, vol. 34, no. 9, pp. 82-88, September 1996

[Pav96e] G. Pavlou, *Computational Specification of the CORBA CMIS Management Broker for the VITAL 1st Trial*, ACTS VITAL project internal document, April 1996

[Pav97a] G. Pavlou, *OSI Systems Management, Internet SNMP and ODP/OMG CORBA as Technologies for Telecommunications Network Management*, in [Aida97], pp. 63-109, IEEE Press, 1997

[Pav97b] G. Pavlou, D. Griffin, *Realizing TMN-like Management Services in TINA*, Journal of Network and System Management (JNSM), Special Issue on TINA, Vol. 5, No. 4, pp. 437-457, Plenum Publishing, 1997 -
also early version in [ISN97], pp. 263-274, Springer, 1997

[Pav97c] G. Pavlou, D. Griffin, *An Evolutionary Approach Towards the Future Integration of IN and TMN*, Interoperable Communications Networks Journal, Special Issue on Service Engineering, Vol. 1, pp. 1-16, Baltzer Science Publishers, 1998 -
also an early version in [ISN95], pp. 12-25, Springer, 1995

[Pav97d] G. Pavlou, *From Protocol-based to Distributed Object-based Management Architectures*, in [DSOM97], pp. 25-40, ISBN 1 86365 3449, 1997

[Pav97e] G. Pavlou, A. Liotta, P. Abbi, S. Ceri, *CMIS/P++: Extensions to CMIS/P for Increased Expressiveness and Efficiency in the Manipulation of Management Information*, to appear in the Proc. of the IEEE INFOCOM'98, to be held in San Francisco, March 1998

[Perr95] G. Perrow, J. Hong, H. Lutfiyya, M. Bauer, *The Abstraction and Modelling of Management Agents*, in [IM-IV], pp. 466-477, Chapman & Hall, 1995

[PREPARE] J. Hall, ed., *Management of Telecommunication Systems and Services - Modelling and Implementing TMN-based Multi-Domain Management*, Springer-Verlag, 1996

[PRISM] K. Berquist, A Besrquist, ed., *Managing Information Highways - The PRISM Book*, Springer-Verlag, 1996

[Proct92] S. Proctor, *An ODP Analysis of OSI Systems Management*, in the Proc. of the TINA'92 Workshop, Narita, Japan, January 1992

[Pyle94] R. Pyle, *Applying Object-Oriented Analysis and Design to the Integration of Network Management Systems*, in [Aida94], pp. 136-159, IEEE Press, 1994

[QUIPU] C. Robbins, S. Kille, *The ISO Development Environment: User's Manual, Version 7.0, Vol 5: QUIPU,* 1991

[Q1200] ITU-T Rec. Q1200 series, *General Series Intelligent Networks Recommendations Structure*, 1992

[Q2761] ITU-T Draft Rec. Q.2761, Broadband ISDN, Signalling System No. 7, B-ISDN User Part, *Functional Description of the B-ISDN User Part (B-ISUP)*, 1995

[Q2931] ITU-T Draft Rec. Q.2931, Broadband ISDN, Digital Subscriber Signalling System No. 2, *User-Network Interface (UNI) Specification for Basic Call/Connection Control*, 1996

[Q3] ITU-T Rec. Q.811, Specifications of Signaling System No. 7, *Q3 Interface*

[Q811] *Lower Layer Protocol Profiles for the Q3 Interface*, 1996

[Q812] *Upper Layer Protocol Profiles for the Q3 Interface*, 1996

[Q751] ITU-T Draft Rec. Q.751, *Signalling System No. 7, Management of the Network Element View for SS7 Networks*, 1996

[Q821] ITU-T Rec. Q.822, Specifications of Signalling System No. 7, *Q3 Interface, Stage 2 and Stage 3 Description for the Q3 Interface - Alarm Surveillance*, 1993

[Q822] ITU-T Rec. Q.822, Specifications of Signalling System No. 7, *Q3 Interface, Stage 1, Stage 2 and Stage 3 Description for the Q3 Interface - Performance Management*, 1994

[Ramo97] A. Ramos, E. Specialski, *The Use of Allomorphism for the Access Control Service in OSI Management Environment*, in [IM-V], pp. 113-126, Chapman & Hall, 1997

[Ranc97] D. Ranc, G. Pavlou, D. Griffin, *A Staged Approach for TMN to TINA Migration*, in [TINA97], 1997

[Reil96] J. Reilly, K. Mourelatou, P. Georgatsos, D. Griffin, G. Pavlou, *Virtual Private Network Management*, in [ICM], pp. 147-187, Crete University Press, 1996

[RMON] S. Walbusser, *Remote Network Monitoring Management Information Base*, IETF RFC 1271, December 1991

[Rose87] M. Rose, D. Cass, *ISO Transport Services on Top of the TCP*, IETF RFC 1006, 1987 -
also in Computer Networks and ISDN Systems, Vol. 12, No. 3, 1986

[Rose88] M. Rose, *ISO Presentation Services over TCP/IP-based Internets*, IETF RFC 1085, 1988

[Rose90] M. Rose, *The Open Book: a Practical Perspective on OSI*, Prentice Hall, 1990

[Rose91] M. Rose, *Network Management is Simple: You Just Need the "Right" Framework*, in [IM-II], pp. 9-25, Elsevier, 1991

[Rumb91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[Rutt94] T. Rutt, ed., *Comparison of the OSI Systems Management, OMG and Internet Management Object Models*, Report of the NMF - X/Open Joint Inter-Domain Management task force, 1994

[Sahi88] V. Sahin, C. Omidyar, T. Bauman, *Telecommunications Management Network (TMN) Architecture and Interworking Designs*, IEEE Journal of Selected Areas in Communications (JSAC), Vol. 6, No. 4, 1988

[Sahi94] V. Sahin, *Telecommunications Management Network: Principles, Models and Applications*, in [Aida94], pp. 72-121, IEEE Press, 1996

[Sartz95] S. Sartzetakis, C. Stathopoulos, V. Kalogeraki, D. Griffin, *Managing the TMN,*, in [ISN95], pp. 200-210, Springer, 1995

[Sartz97] S. Sartzetakis, P. Georgatsos, G. Pavlou, D. Griffin, *Unified Fault, Resource Management and Control in ATM-based IBCN*, in [IM-V], pp. 262-274, Chapman & Hall, 1997

[Schad96] A. Schade, P. Trommler, *A CORBA-based Model for Distributed Applications Management*, in [DSOM96], 4.1/1-10, 1996

[Schmi97] D. Schmidt, A. Gokhale, T. Harrison, G. Parulkar, *A High-Performance End System Architecture for Real-Time CORBA*, IEEE Communications, Special Issue on Distributed Object Computing, vol. 35, no. 2, pp. 72-77, February 1997

[Seitz97] J. Seitz, *Meta Managed Objects*, in [IM-V], pp. 650-660, Chapman & Hall, 1997

[Shrew95] K. Shrewsbury, *An Introduction to TMN*, Journal of Network and System Management (JNSM), Vol. 3, No. 1, pp. 13-38, Plenum Publishing, 1995

[Sidor95] D. Sidor, *Managing Telecommunications Networks Using TMN Interface Standards*, IEEE Communications, pp. 54-60, March 1995

[Sidou95] D. Sidou, S. Mazziotta, R. Eberhardt, *TIMS: a TMN-based Information Model Simulator and Principles*, in [DSOM95], pp. 8.1/1-10, 1995 -
also in [IM-V] as *Design and Testing of Information Models in a Virtual Environment*, pp. 461-472, Chapman & Hall, 1997

[Slom89] M. Sloman, J. Moffett, *Domain Management for Distributed Systems*, in [IM-I], pp. 505-516, Elsevier, 1989

[Slom94] M. Sloman, ed., *Network and Distributed Systems Management*, Addison-Wesley, 1994

[Slum89] C. Sluman, *A Tutorial on OSI Management*, Computer Networks and ISDN Systems, Vol. 17, pp. 270-278, 1989

[SMF] ITU-T Rec. X.730-X.750, Information Technology - Open Systems Interconnection, *Systems Management Functions*

[X730] *Object Management Function,* 1992

[X731] *State Management Function,* 1992

[X732] *Attributes for Representing Relationships,* 1992

[X733] *Alarm Reporting Function,* 1992

[X734] *Event Report Management Function,* 1992

[X735] *Log Control Function,* 1992

[X736] *Security Alarm Reporting Function,* 1992

[X737] *Confidence and Diagnostic Testing,* 1992

[X738] *Summarisation Function,* 1993

[X739] *Metric Objects and Attributes,* 1993

[X740] *Security Audit Trail Function,* 1992

[X741] *Objects and Attributes for Access Control,* 1995

[X742] *Accounting Metering Function,* 1993

[X743] *Time Management Function*, 1995

[X744] *Software Management Function,* 1995

[X745] *Test Management Function,* 1993

[X746] *Scheduling Function,* 1995

[X748] *Response Time Monitoring Function,* 1996

[X749] *Domain and Policy Management Function*, 1996

[X750] *Management Knowledge Management Function,* 1995

[SMI] ITU-T Rec. X.720-X.725, Information Technology - Open Systems Interconnection, *Structure of Management Information*

[X720] *Management Information Model (MIM)*, 1991

[X721] *Definition of Management Information (DMI)*, 1992

[X722] *Guidelines for the Definition of Managed Objects (GDMO)*, 1992

[X725] *General Relationship Model (GRM)*, 1995

[Smith90] C. Smith, D. Milham, C. Mulcahy, *OSI Systems Management*, BT Technology Journal, pp. 27-38, April 1990

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

[Spiv89] J. Spivey, *The Z Notation: a Reference Manual*, Prentice Hall, 1989

[Stal93] W. Stallings, *SNMP, SNMPv2 and CMIP: The Practical Guide to Network-Management Standards*, Addison-Wesley, 1993

[Stath93] C. Stathopoulos, *Location Transparency Support via the X.500 Directory in OSIMIS - Architecture, Configuration and User's Manual Version 1.0*, RACE ICM Project Internal Document, 1993

[Stath95] C. Stathopoulos, D. Griffin. S. Sartzetakis, *Handling the Distribution of Information in the TMN*, in [IM-IV], Chapman & Hall, 1995

[Strau86] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986

[Strut89] C. Strutt, D. Shurtleff, *Architecture of an Integrated, Extensible Enterprise Management Director*, in [IM-I], pp. 61-72, Elsevier, 1989

[Strut94] C. Strutt, M.W. Sylor, *DEC's Enterprise Management Architecture*, in [Slom94], pp. 581-604, Addison-Wesley, 1994

[Sun87] Sun Microsystems Inc., *XDR: External Data Representation Standard*, IETF RFC 1014, 1987

[Sun88] Sun Microsystems Inc., *Remote Procedure Call Protocol Specification*, IETF RFC 1057, 1988

[Sun96] Sun Microsystems Inc.: A. Ken, J. Gosling, *The Java Programming Language*, Addison-Wesley, 1996

[Sylor89] M. Sylor, *Guidelines for Structuring Manageable Entities*, in [IM-I], pp. 169-183, Elsevier, 1989

[Sylor93] M. Sylor, *Junction Objects: A Solution to a Problem in Naming and Locating OSI Managed Objects*, in [IM-III], pp. 161-173, Elsevier, 1993

[Tanen96] A. Tanenbaum, *Computer Networks*, 3rd Edition, Prentice Hall, 1996

[Tin95] T. Tin, G. Pavlou, R. Shi, *Tcl-MCMIS: Interpreted Management Access Facilities*, in [DSOM95], pp. 5.3/1-11, 1995

[TINA] M. Chapman, F. Dupuy, G. Nilsson, *An Overview of the Telecommunications Information Networking Architecture*, in [TINA95], pp. 1-12, 1995

[TINA95] Proc. of the *Telecommunications Information Network Architecture (TINA'95)* International Workshop, Melbourne, Australia, February 1995

[TINA96] K. Birman, J. Claus, ed., Proc. of the *Telecommunications Information Network Architecture (TINA'96)* International Conference, VDE-Verlag, Heidelberg, Germany, September 1996

[TINA97] Proc. of the *Telecommunications Information Network Architecture (TINA'97)* International Conference, Santiago, Chile, November 1997

[Tirop97] T. Tiropanis, D. Lewis, A. Richter, R. Shi, *A Service Engineering Approach to Inter-Domain TMN System Development*, in [IM-V], pp. 165-176, Chapman & Hall, 1997

[TPMIB] ISO 15123-1, *Common Management Information for the Lower Layer Profiles - Part 1: Transport Layer Management Information*, 1996

[Trate96] F. Trate, *VITAL's Trader Specifications*, Alcatel Alsthom Recherche, Document Ref. ULT/LL/96/474, 1996

[Tschi93] M. Tschichholz, U. Meyer zu Natrup, S, Waberoth, *A Service for Administering Management Information - VPN Inter-Domain Management Support Using X.500 and X.700*, in [IM-III], pp. 137-148, Elsevier, 1993 -
also slightly different version by M. Tschichholz, W. Donnely, *The PREPARE Management Information Service*, in [ISN93], 1993

[Tuck94] J. Tucker, *OSI Structure of Management Information*, in [Slom94], pp. 95-130, Addison-Wesley, 1994

[Vass95] N. Vassila, G. Knight, *Introducing Active Managed Objects for Effective and Autonomous Distributed Management*, in [ISN95], pp. 415-429, Springer, 1995

[Vass97] N. Vassila, G. Pavlou, G. Knight, *Active Objects in TMN*, in [IM-V], pp. 139-150, Chapman & Hall, 1997

[Vino97] S. Vinoski, *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*, IEEE Communications, Special Issue on Distributed Object Computing, vol. 35, no. 2, pp. 46-55, February 1997

[Whit97] R.B. Whitner, *Designing Scaleable Applications Using CORBA*, in [IM-V], pp. 503-514, Chapman & Hall, 1997

[Wilb87] S. Wilbur, B. Bacarisse, *Building Distributed Systems with Remote Procedure Call*, IEE Software Engineering Journal, Vol. 2, No. 5, pp. 148-159, 1987

[Wirth71] N. Wirth, *Program Development by Stepwise Refinement*, Communications of the ACM, Vol. 14, No. 4, pp. 221-227, 1971

[Wirth75] N. Wirth, *An Assessment of the Programming Language Pascal*, IEEE Transactions on Software Engineering, Vol. 1, pp. 192-198, 1975

[Wirth82] N. Wirth, *Programming in Modula-2*, Springer-Verlag, 1982

[XMP] X/Open, *Systems: Management: Management Protocols API (XMP)*, 1992 -
also published as ISBN 1-85912-027-X in 1994

[XOM] X/Open, *OSI-Abstract-Data Manipulation API (XOM)*, 1992 - also published as ISBN 1-85912-008-3 in 1994

[X200] ITU-T Rec. X.200, Information Technology - Open Systems Interconnection, *OSI Reference Model*, 1988

[X208] ITU-T Rec. X.208, Information Technology - Open Systems Interconnection, *Specification of Abstract Syntax Notation One (ASN.1)*, 1988

[X209] ITU-T Rec. X.209, Information Technology - Open Systems Interconnection, *Specification of Basic Encoding Rules (BER) for ASN.1*, 1988

[X217] ITU-T Rec. X.217, Information Technology - Open Systems Interconnection, *Service Definition for the Association Control Service Element (ACSE)*, 1988

[X219] ITU-T Rec. X.219, Information Technology - Open Systems Interconnection, *Remote Operations: Model, Notation and Service Definitions*, 1988

[X500] ITU-T Rec. X.500, Information Technology - Open Systems Interconnection, *The Directory: Overview of Concepts, Models and Service*, 1993

[X511] ITU-T Rec. X.511, Information Technology - Open Systems Interconnection, *The Directory: Abstract Service Definition*, 1993

[X700] ITU-T Rec. X.700, Information Technology - Open Systems Interconnection, *OSI Management Framework*, 1992

[X701] ITU-T Rec. X.701, Information Technology - Open Systems Interconnection, *Systems Management Overview*, 1992

[X703] ITU-T Rec. X.703, Information Technology - Open Systems Interconnection, *Open Distributed Management Architecture*, 1997

[X710] ITU-T Rec. X.710, Information Technology - Open Systems Interconnection, *Common Management Information Service Definition (CMIS) - Version 2*, 1991

[X711] ITU-T Rec. X.711, Information Technology - Open Systems Interconnection, *Common Management Information Protocol Specification (CMIP) - Version 2*, 1991

[X800] ITU-T Rec. X.800, Information Technology - Open Systems Interconnection, *OSI Security Architecture*, 1991

[Yemi91] Y. Yemini, G. Goldszmidt, S. Yemini, *Network Management by Delegation*, in [IM-II], pp. 95-107, Elsevier, 1991

[Yemi93] Y. Yemini, *The OSI Network Management Model*, IEEE Communications, pp. 20-29, May 1993

[Zatti94] S. Zatti, *Name Management and Directory Services*, in [Slom94], pp. 247-272, Addison-Wesley, 1994

[Z100] ITU-T Rec. Z.100, *Specification and Description Language SDL*, 1989