X/Open Guide

Systems Management: Managed Object Guide (XMOG)

X/Open Company Ltd.

/

X/Open Guide

Systems Management: Managed Object Guide (XMOG)

ISBN: 1-85912-006-7

X/Open Document Number: G302

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Figures

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and allows users to move between systems with a minimum of retraining.

The components of the Common Applications Environment are defined in X/Open CAE Specifications. These contain, among other things, an evolving portfolio of practical application programming interfaces (APIs), which significantly enhance portability of application programs at the source code level, and definitions of, and references to, protocols and protocol profiles, which significantly enhance the interoperability of applications.

The X/Open CAE Specifications are supported by an extensive set of conformance tests and a distinct X/Open trademark - the XPG brand - that is licensed by X/Open and may be carried only on products that comply with the X/Open CAE Specifications.

The XPG brand, when associated with a vendor's product, communicates clearly and unambiguously to a procurer that the software bearing the brand correctly implements the corresponding X/Open CAE Specifications. Users specifying XPG-conformance in their procurements are therefore certain that the branded products they buy conform to the CAE Specifications.

X/Open is primarily concerned with the selection and adoption of standards. The policy is to use formal approved *de jure* standards, where they exist, and to adopt widely supported *de facto* standards in other cases.

Where formal standards do not exist, it is X/Open policy to work closely with standards development organisations to assist in the creation of formal standards covering the needed functions, and to make its own work freely available to such organisations. Additionally, X/Open has a commitment to align its definitions with formal approved standards.

X/Open Specifications

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the long-life specifications that form the basis for conformant and branded X/Open systems. They are intended to be used widely within the industry for product development and procurement purposes.

Developers who base their products on a current CAE Specification can be sure that either the current specification or an upwards-compatible version of it will be referenced by a future XPG brand (if not referenced already), and that a variety of compatible, XPG-branded systems capable of hosting their products will be available, either immediately or in the near future.

CAE Specifications are not published to coincide with the launch of a particular XPG brand, but are published as soon as they are developed. By providing access to its specifications in this way, X/Open makes it possible for products that conform to the CAE (and hence are eligible for a future XPG brand) to be developed as soon as practicable, enhancing the value of the XPG brand as a procurement aid to users.

- *Preliminary Specifications*

  These are specifications, usually addressing an emerging area of technology, and consequently not yet supported by a base of conformant product implementations, that are released in a controlled manner for the purpose of validation through practical implementation or prototyping. A Preliminary Specification is not a ‘‘draft’’ specification. Indeed, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE Specification.

  Preliminary Specifications are analogous with the ‘‘trial-use’’ standards issued by formal standards organisations, and product development teams are intended to develop products on the basis of them. However, because of the nature of the technology that a Preliminary Specification is addressing, it is untried in practice and may therefore change before being published as a CAE Specification. In such a case the CAE Specification will be made as upwards-compatible as possible with the corresponding Preliminary Specification, but complete upwards-compatibility in all cases is not guaranteed.

In addition, X/Open periodically publishes:

- *Snapshots*

  Snapshots are ‘‘draft’’ documents, which provide a mechanism for X/Open to disseminate information on its current direction and thinking to an interested audience, in advance of formal publication, with a view to soliciting feedback and comment.

  A Snapshot represents the interim results of an X/Open technical activity. Although at the time of publication X/Open intends to progress the activity towards publication of an X/Open Preliminary or CAE Specification, X/Open is a consensus organisation, and makes no commitment regarding publication.

  Similarly, a Snapshot does not represent any commitment by any X/Open member to make any specific products available.

X/Open Guides

X/Open Guides provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant.

X/Open Guides are not normative, and should not be referenced for purposes of specifying or claiming X/Open-conformance.

This Document

This document is Guide (see above). It is one of several documents within X/Open's Systems Management programme (XSM).

The XSM programme addresses distributed systems management. The primary requirement is to promote the development of management software that allows an administrator to manage a network of heterogeneous systems as a single logical system.

The XSM programme is concerned with the definition of those specifications necessary for the implementation of distributed management systems. In order to meet the goal of interoperability it is necessary for differing implementations to share both a common means of transferring information *and* a common understanding of that information.

The first of these requirements is met by defining communications protocol profiles, and the second by the definition of a structure for management information. XSM makes use of object-oriented techniques in its specifications primarily to facilitate this second requirement. Within XSM, resources to be managed are modelled as one or more managed objects.

The aim of this document is to introduce the essential nature of managed objects, and provide guidelines for identifying and selecting suitable managed objects, and the necessary framework for defining them. It discusses the managed object development process and their definition, and explores the issues involved in registering them and on conformance testing.

Trade Marks

Acknowledgements

X/Open gratefully acknowledges the contribution by the Open Software Foundation of the IDL examples in Section A.3. This contribution represents work-in-progress, and is intended for use only as an example.

X/Open gratefully acknowledges the contribution by the Network Management Forum ISO/Internet Management Coexistence (IIMC) working group of the GDMO examples in Section A.2. This contribution represents work-in-progress, and is intended for use only as an example.

Referenced Documents

The following documents are referenced in this guide:

CMD
: RFC1212, Concise MIB Definition, M. Rose, & K. McCloghrie.

COIW
: X/Open Guide, November 1992, ISO/CCITT and Internet management: Coexistence and Interworking Strategy (ISBN: 1-872630-67-7, G211)

CORBA
: X/Open Preliminary Specification, February 1993, The Common Object Request Broker: Architecture and Specification (ISBN: 1-872630-90-1, P210).

GDMO
: ISO/IEC 10165-4: 1992 Information technology — Open Systems Interconnection — Structure of management information — Part 4: Guidelines for the definition of managed objects. This is equivalent to CCITT X.722.

GRM
: ISO draft: General Relationship Model, ISO SC21/WG4/N6041.

MIB
: RFC 1214, April 1991 — OSI Internet Management: Management Information Base, L. Labarre

MIB-II
: RFC 1213, March 1991 — Management Information Base for the Network Management of TCP/IP-based Internets (MIB-II), M.Rose, & K. McCloghrie

MIM
: ISO/IEC 10165-1: 1992 Information technology — Open Systems Interconnection — Structure of management information — Part 1: Management information model. This is equivalent to CCITT X.720.

MPMO
: OSI Network Management Forum, Technical Report 102: Modelling Principles for Managed Objects.

MWD
: Schlaer S & Mellor SJ, (1988), Object-Oriented Systems Analysis: Modelling the World in Data, Prentice Hall, Englewood Cliffs, New Jersey 07632.

OMAG
: Object Management Architecture Guide 1.0, Chapter 4, OMG TC Document 90.9.1, Object Management Group Inc.

OOA
: P Coad & E Yourdon, (1991), Object-Oriented Analysis, Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632 (ISBN 0-13-629981-4).

OOD
: P Coad & E Yourdon, (1991), Object-Oriented Design, Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632 (ISBN 0-13-629981-4).

OODM
P. Arnold, S. Bodoff, et al, (1991), An Evaluation of Five Object-Oriented Development Methods, HP Laboratories, Bristol, HPL-91-52 (compares the Booch, Buhr, HOOD, Rumbaugh and Wirfs-Brock methods for object-oriented analysis and/or design).

OODRM
ANSI Object-Oriented Database Task Group — Object Data Management Reference Model, Document No. OODB89-01R8, 17 September 1991.

OOSD
Colbert E, (1989), Object-Oriented Software Development: A Practical Approach to Object-Oriented Development, TRI-Ada '89, Pittsburgh, PA, ACM SIGAda.

PS
X/Open Snapshot, 1991, Systems Management: Problem Statement, (XO/SNAP/91/010, S110).

SMI
RFC 1155, Structure and Identification of Management Information for TCP/IP-based Internets (SMI), M. Rose, & K. McCloghrie

SMIv2
RFC 1442, Structure on Management Information for version 2 of the Simple Network Management Protocol (SNMPv2), J. Case, K. McCloghrie, M. Rose, & S. Waldbusser.

SNMP
RFC 1157, A Simple Network Management Protocol (SNMP), J. Case, M. Fedor, M. Schoffstall, & J. Davin.

SNMPv2
RFC 1448, Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2), J. Case, K. McCloghrie, M. Rose, & S.Waldbusser.

SPM
Schlaer S & Mellor SJ, (1988), Object-Oriented Systems Analysis: State and Process Models, Prentice Hall, Englewood Cliffs, New Jersey 07632.

XRM
X/Open Guide, August 1993, Systems Management: Reference Model (ISBN: 1-85912-05-9, G207).

Chapter 1

Introduction

The X/Open Systems Management Programme is developing a series of specifications and other documents in the area of distributed systems management. It is intended to satisfy several high-level system requirements:

Portability	The ability to make software on managed and managing systems portable in source code form between different vendors' systems by extending the X/Open Common Applications Environment (CAE). An important additional aspect of portability is the "portability" of human administrators, that is, the ability of an administrator to move between systems and benefit from a consistent user interface.
Interoperability	The ability of management systems, and components of such systems from different vendors, to interwork, thus allowing a network of heterogeneous systems to be managed as a single system.
Transparency	The ability to administer Resources without the need to be explicitly aware of their location or details of their implementation.
Extensibility	The ability to extend the scope and capabilities of the management system and to implement different management policies as required. This includes the ability to make use of new communications protocols.
Robustness	The ability of the management system to provide integrity and the necessary levels of security and reliability.

The following requirements relate to the form of the interfaces that will be provided to access the management functionality:

Ease of Use	The services and APIs should be simple to use, consistent with the complexity of the underlying functionality.
Consistency	Wherever appropriate, stylistic inconsistency should be avoided in specification of interfaces.

The X/Open Systems Management Programme is defined in terms of a suite of documents that, taken together, will describe all the components needed to achieve the goals listed above.

The first of these documents is the X/Open Systems Management Problem Statement (reference **PS**). The Problem Statement provides an overview of the problem and a review of current activities.

The X/Open Systems Management Reference Model (reference **XRM**) builds on the Problem Statement, providing a framework in which the various components of the solution can be identified. The individual components will be defined in subsequent documents.

The Reference Model is based on the use of object-oriented specification techniques. This in no way requires an implementation to use object-oriented technology. Object-oriented techniques have been adopted in this area by several other bodies, including vendors, standards bodies, and other industry consortia.

1.1 Intended Audience

Fundamental to the approach adopted in the X/Open Systems Management Programme is the use of *Managed Objects* to represent the real Resources that are being managed. The purpose of this document is to explore many of the issues relating to the development of Managed Objects. The Managed Object Guide provides a combination of introductory, reference, and tutorial material intended for the following audience:

- Implementers of Managed Objects.
- Users of management systems who wish a better understanding of the underlying concepts.
- End-users who require to create Managed Object objects derived from pre-existing objects.

Chapter 2

Overview

The X/Open Systems Management Programme (XSM) is concerned with the definition of those specifications necessary for the implementation of distributed management systems. In order to meet the goal of interoperability it is necessary for differing implementations to share both a common means of transferring information *and* a common understanding of that information.

The first of these requirements is met by defining communications protocol profiles, and the second by the definition of a structure for management information. XSM makes use of object-oriented techniques in its specifications primarily to facilitate this latter requirement. Within XSM, Resources to be managed are modelled as one or more Managed Objects.

2.1 The Managed Object Development Process

In order for this technique to be effective, all implementations must share a common means of defining Managed Objects, and a common means of encoding that information for transmission to other systems. Finally, interoperable implementations must also share a common understanding of the *meaning* of the information, such that, when a system requests another system to perform a particular operation, the desired changes to the environment are actually carried out. The existence of these specifications allows true interoperability between management systems.

The aim of this document is to provide guidelines for identifying and selecting Managed Objects, and the necessary framework for defining them.

In order to produce interoperable Managed Objects, it is necessary to perform a process consisting of several steps. The sequence of steps is shown in Figure 2-1 on page 5.

The steps within the process can be summarised as follows:

Identification
: One of the most critical steps in the process is that of identifying the set of Managed Objects that will be used to model the Resource to be managed. The success of this identification is crucial to the simplicity and usability of the interface that is provided. This topic is covered in Chapter 3.

Search
: Once a set of Managed Objects has been identified, each must be defined. Before this stage is undertaken, it is generally wise to perform a search of existing Managed Object definitions in order to determine whether suitable objects have already been defined.

Refinement
: If a suitable pre-existing Managed Object definition is identified, it may be necessary to make some modifications to ’’refine ’’ the definition.

Definition
: If no search has been performed, or if no suitable existing Managed Object definitions have been identified, then Managed Object definitions must be developed. This process is covered in Chapter 4.

Registration and Publication
: In order for the Managed Objects to be used by multiple systems, their definitions must be made available to those systems. Key aspects of this process are the registration of object definitions with registration authorities and the publication of definitions so that they can be used by systems administrators and Managed Object definers. Registration is covered in Chapter 7.

Cataloguing
: In order to encourage their use, published object definitions should be readily available. Thus, administrators and Managed Object definers should be able to refer to catalogues of Managed Objects when searching for definitions that can model a Resource they wish to model.

Implementation
: Implementation of Managed Objects generally involves the production of software that provides the functionality contained within the Managed Object definitions. Details of implementation techniques are dependent on the precise technology being used and are beyond the scope of this document.

Conformance Testing
: Conformance testing is a key part of the process if interoperability is to be achieved. Some means of verifying that different implementations of a Managed Object definition are equivalent is essential in order for full functionality to be achieved. An administrator initiating a management action must have confidence that the defined behaviour will occur. This is covered in Chapter 9.

Use
: Once the identified Managed Objects have been implemented, they are available for use by management systems.

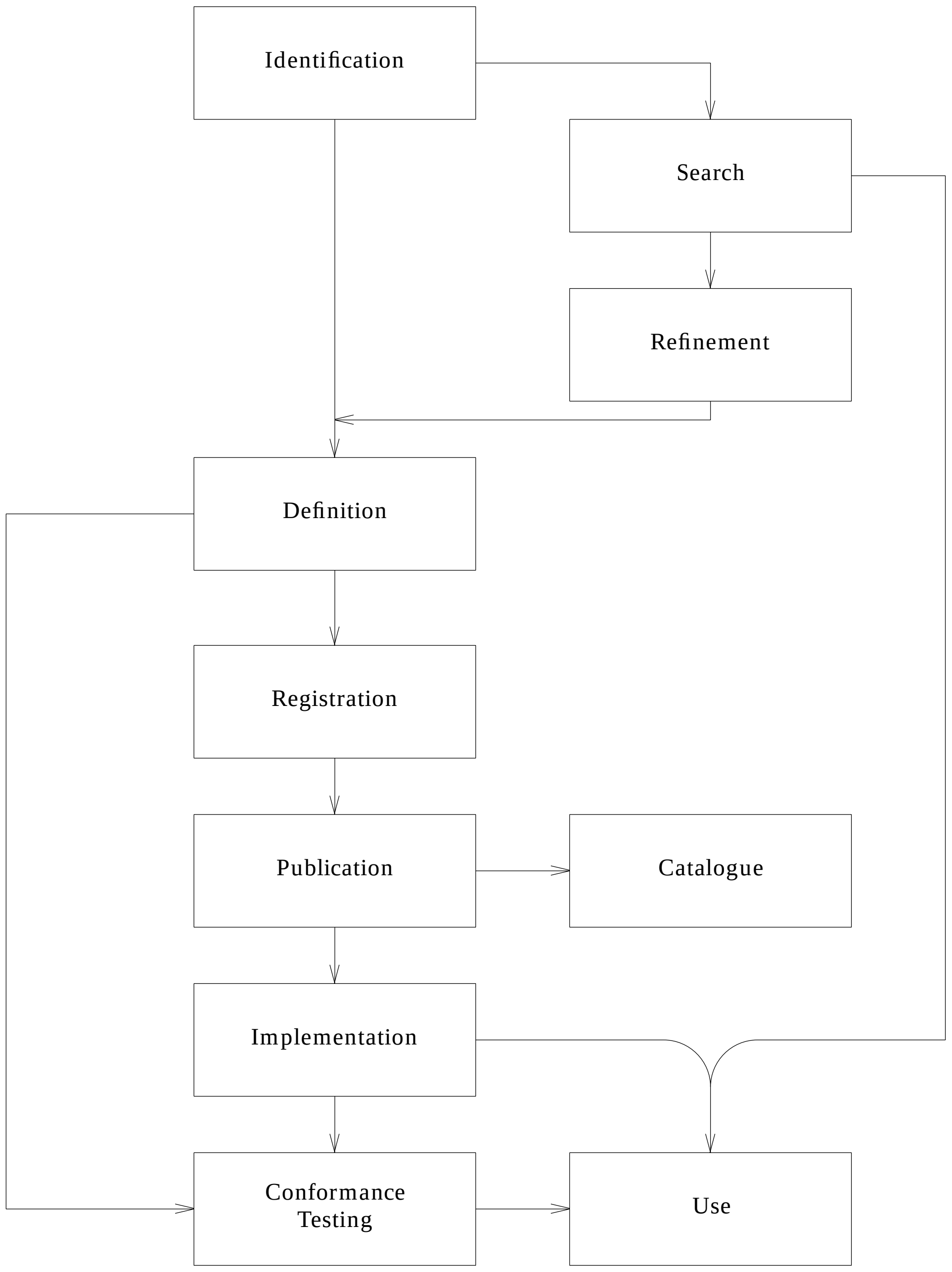

Figure 2-1 Managed Object Development Process

At first sight, this process may seem long and complicated. In practice, although this process will never become mechanical, as the global catalogue of Managed Objects grows, more and more requirements will be met from reuse or refinement of existing definitions. The most common activity will probably be the refinement of an existing generic Managed Object definition, (for example, a user Managed Object), into a specific definition customised for a user's environment. This may simply require the definition of suitable management policy or may involve some measure of modification in order to provide additional functionality that represent some locally-defined additions to the environment.

2.2 Object Models

The object-oriented approach has been used by a number of groups defining network and systems management standards. This has resulted in the definition of a number of object models and consequently, one must take care to use the appropriate one for the task at hand. These models have a number of common features, but are often not directly compatible. They differ not only in the protocols and encodings they use but also in some of the basic concepts.

The management object models of primary interest to X/Open are those that describe the facilities needed for network management for OSI and Internet networks. These are primarily intended for network management, but can be naturally extended to cover distributed systems management.

In addition to the specifically management-oriented models, there are also general-purpose object models, notably that defined by the Object Management Group (OMG). The primary focus of the OMG is to to promote easier application integration within a distributed environment. Recognising that distributed management systems are essentially no different to any other distributed applications, such general-purpose technology is increasingly being used for management.

The OMG has defined a general object-oriented approach that can be used to model network and systems management. OMG has issued two important documents; one provides a general definition of an object model (reference **OMG**), and the other defines an *Object Request Broker* (reference **CORBA**), which provides a general mechanism by which objects transparently make requests and receive responses. The ORB is intended to provide interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

Chapter 3

Identification of Managed Objects

A Managed Object is the XSM view of a Resource within the X/Open Systems Management environment that is subject to management. This would include both physical Resources, such as printers or disks, and logical Resources, such as an application program. Thus, a Managed Object is the abstract view of such a Resource that represents its properties as seen by (and for the purposes of) management.

An important term in systems management is **Management Information Base (MIB)**. Within ISO, the term MIB is used to describe the set of Managed Objects within a system. However, within the Internet community, the term MIB has a more concrete definition, meaning a specific set of general definitions for the purposes of network management; thus an Internet system will implement one or more specific MIBs.

The selection of the set of Managed Objects is a very critical process in the provision of a fully-featured management system. An important goal of XSM is extensibility, and a major aspect in achieving this goal is the ability to define new Managed Objects. While a basic set of Managed Objects will be identified and defined within XSM, it is intended that this will primarily provide a set of prototypical Managed Objects that will be refined to produce that set of Managed Object definitions specific to a particular installation or environment.

Having stated that the selection of Managed Objects is of crucial importance, it must also be pointed out that there are no simple rules that can be used in order to accomplish this. In general, there will be several different ways in which any given Resource can be broken down into one or more Managed Objects. This breakdown will significantly influence the management interface available to manage the Resource.

In many cases, a good starting point in the identification process will be the consideration of the Management Tasks that need to be supported. Such tasks, which represent the Administrator's view of the operation that he wishes to perform, give a high-level view of the low-level Managed Objects that are needed in order to support the operation. They tend to comprise relatively *high-level* concepts such as *adding a user*, or modifying a mail system routing. However, the capabilities of the Resource being modelled must also be considered. Thus, there are two important considerations when attempting to define the Managed Objects required to perform a Management Task:

- Management Requirements
- Resource Functionality / Capabilities.

The former considers the Management Tasks that need to be supported, while the latter considers the physical Resource and the restrictions it imposes on the capabilities that can be modelled by the Managed Object, and made available to management systems. For example, a management requirement may exist for a ''printer-out-of-paper'' notification, but a particular Resource (for example, a printer) may not be able to generate such a notification.

When considering the management of a particular Resource for which it is desired to define Management Tasks and Managed Objects, the two views of management requirements *versus* Resource functionality/capabilities represent the two extremes of the top-down and bottom-up approach to the problem. These must be used to guide an iterative approach to the definition of Managed Objects in order to ensure that the following key requirements are met:

- the functionality provided by the Managed Objects is sufficient to support the Management Tasks that must be performed

- the granularity of the Managed Objects is at the right level to permit efficient implementation.

These issues are addressed in the following two Chapters.

In addition, there may be other objectives in defining Managed Objects. For example:

- a Managed Object definition may be required in order to be provide the basis from which other Managed Objects may be derived (for example, an X.25 virtual circuit object may be defined in order to derive from it permanent and switched virtual circuit Managed Object definitions).
- a Managed Object may be defined in order to provide a facility required by many other Managed Objects (for example, an event forwarding discriminator may be used by many objects that generate events).
- access control and security considerations may impose restrictions on how Managed Objects are defined, and/or may require the definition of additional Managed Objects.

3.1 Management Tasks

Systems management is performed by people, and therefore the tasks available have to be intuitive to Administrators and reflect what they do from day to day. To be intuitive, the tasks are likely to represent relatively simple operations on relatively complex Resources. While the implementation of these tasks is in terms of simple operations on simple Managed Objects, the purpose of the task is to shield the Administrator from most of the routine detail, such as the sequencing of the individual operations.

By way of example, when an Administrator wishes to add a user to the system, they need to specify the various properties of that user, such as a login-name, user-Id, group membership information, home directory, login shell, etc. In many cases some of these properties may be provided to him by the system in the form of default values which reflect local policy. The order in which the operations involving these properties is performed is clearly significant. For instance, when a user is added to a system, the userid has to be created before a correct ownership can be given to the assigned directory. However, the precise ordering is not important to the Administrator performing the task. The Administrator is simply concerned with the end result of the task and relies on its implementor to perform the operations in the correct order.

3.2 Managed Objects

It follows from the discussion of Management Tasks that there need not be a simple one-to-one mapping between the Resource being managed and the Managed Objects that represent it.

Pursuing the example in the previous section, the process of adding a user involves the manipulation of several different Resources, userids, the filesystem, security information, etc. While all the properties considered by the Administrator when he performs the task could simply be assigned to a single user Managed Object, there may be excellent reasons for associating them with a number of different, simpler Managed Objects. For example, a user Managed Object might consist of the most basic information, and be supplemented by additional Managed Objects that encapsulate:

- aspects of the user's interaction with other Resources (such as the print and mail systems)
- aspects of management policy
- aspects of shared management functions
- aspects of the user's interaction with Managed Objects.

The simplest guideline that can be established is the principle of maintaining a practical set of the simplest Managed Objects. If the Managed Object definition appears to be becoming too complicated, it is very probably wrong and a new breakdown of the Resource's properties should be considered.

One important aspect that should be considered is the definition of common Management Services. If a certain functionality is required by many different Managed Objects, it could be provided as a common service and defined once only. As well as simplifying the Managed Object definition, this also has the benefit of ensuring that shared functionality is defined consistently.

As has been stated above, there are no simple rules that can be used to make the selection of the right Managed Objects a mechanical process. A guide to the process of object modelling can be found in the Network Management Forum Technical Report 102 (reference **MPMO**). There are many books and papers published on this subject, and by way of introduction, a few (**OODM**, **OOD**, **OOSD**, **MWD**, **SPM**) are included in the list of **Referenced Documents** in this Guide.

Chapter 4

Definition of Managed Objects

This Chapter documents the capabilities provided by the OSI, Internet, and OMG object models. It is beyond the scope of this document to provide full tutorial information, but it is intended to provide an overview of the fundamental concepts. The material provided here is supplemented by the provision of a common example in Appendix A.

4.1 Introduction

The object-oriented approach is now widely supported and has been adopted by a number of groups to model network and systems management. This approach was pioneered by the ISO Network Management standards, and was subsequently adopted by the Internet community. The example of this approach was adopted for Systems Management standardisation by the IEEE P1003.7 POSIX Systems Administration working group. More recently the Object Management Group has defined a general-purpose, object-oriented model for application integration. Although the focus of the OMG was not specifically targetted towards Systems Management, the OMG technology has been widely adopted as the basis of Systems Management products.

As the various approaches have evolved in different ways, it is inevitable that they have developed differing terminologies to describe the properties of the objects defined according to their respective object models. Much of this differing terminology reflects the different approach that each model employs.

By way of example, The OSI approach can be characterised as being *attribute-rich*, with the properties of the underlying Resource being represented by attributes on which explicit set and get operations can be performed. The OMG model is, in contrast, *method-rich*, where the interface to the Resource is represented by a set of methods which manipulate those properties.

Much energy can be (and has been) expended on arguing the relative merits of the differing approaches, and at times this can obscure the original issue, namely the problem that the particular approach is being employed to solve. This Chapter describes the characteristics of the object models that will be used by the X/Open Systems Management programme.

4.1.1 The OMG Approach

In the OMG approach, an object model has been developed which can be used to implement a variety of applications and services. The model supports a strongly-typed definition language which separates interface from implementation, thus supporting a high degree of encapsulation through multiple implementations. The model provides a client-server view of applications and services; that is, a client sends requests to a server in which the object has been manifested, the object executes the request, and returns the results back to the client. The interface definition language (IDL) allows developers to define their objects in a manner that is not only more consistent with their application's view of the managed data, but also consistent with object-oriented programming languages, and consistent with object-oriented databases.

The architecture supporting the OMG object model provides secure, distributed access to the data encapsulated in the objects. The object is accessed through a opaque object referent, or handle, and its location is transparent to the client making the request. The infrastructure, consisting of an Object Request Broker (ORB) and an Object Adaptor (OA) provide not only a transport, but also a consistent method invocation paradigm. This simplifies object design while enabling portability, interoperability, and building objects which encapsulate the architecture of the application tasks.

The OMG model defines a set of common object services. These services are themselves encapsulated as objects within the OMG model, so they can be defined in the same definition language used by application developers. (There are some exceptions, such as persistent object stores.) Over time, more abstract services (higher levels of functionality) will be defined including profiles of services oriented to meet specific application areas. Systems management is such an application area; that is systems management in an OMG environment would consist of management applications built from application programs, Management Services, common object services, and management objects. The management objects would encapsulate potentially complex Management Tasks, using the underlying Management Services, common object services, and other objects. These other objects include both Managed Objects (a managed Resource encapsulated by an object) and OMG objects from non-management applications.

4.1.2 The ISO Approach

ISO has used an object-oriented approach in the ISO/IEC 10165, Structure of Management Information standard. This is a multi-part standard, and Parts 1 (Management Information Model, reference **MIM**) and 4 (Guidelines for the Definition of Managed Objects, (reference **GDMO**) are of particular interest to X/Open. The GDMO is discussed later in this document.

The MIM and GDMO have been used by other groups to define Managed Objects. For example, they have been used by the Network Management Forum (NMF) in a series of interrelated documents; each document describes either an aspect of Forum interoperable network management or provides a framework on which other documents are based. The NMF adheres to ISO standards wherever possible, and otherwise uses *snapshots* based on existing ISO drafts to achieve full coverage in its specifications. It refines the standards by adding implementation agreements to provide a complete specification for multi-developer, interoperable implementations. The NMF Release 1 specifications were based on a snapshot of ISO CD-level standards. The Release 1.1 specifications were updated to use the DIS GDMO templates. In conjunction with other organisations including X/Open, the NMF has defined a comprehensive set of Network Management specifications, OMNI*Point* 1, which are based on the ISO IS-level standards.

4.1.3 The Internet Approach

The Internet Advisory Board has also adopted an object-oriented approach in defining network management for the Internet. It recommends that all IP and TCP implementations be network-manageable. This implies implementation of the Internet Management Information Base II (MIB-II) described in RFC1213 (see reference **MIB-II**). There are currently two versions of Internet management which can be used in conjunction with MIB-II. Version 1 of the Simple Network management protocol (SNMP) is defined by RFC 1157 (see reference **SNMP**), the corresponding Structure of Management Information is defined by RFC 1155 (see reference **SMI**) and RFC 1212. Version 2 of SNMP (SNMPv2, currently a proposed draft Internet standard) is defined by RFC 1448 (see reference **SNMPv2**); the corresponding SMI is defined by RFC 1442 (see reference **SMIv2**).

4.1.4 The X/Open Approach

The purpose of a Structure of Management Information (SMI) Model is to give structure to the management information conveyed externally by systems management protocols and to model the management aspects of the related Resources. The X/Open Systems Management Model is expressed in terms of object-oriented terminology, where systems management Resources are represented by objects. Systems management objects (Managed Objects) are abstractions of the data processing and data communications Resources for the purpose of management.

The distinction between the Managed Object as visible to management and the Resources that it represents may be described by saying that the attributes, operations and notifications are visible to management at the Managed Object boundary, whereas the internal functioning of the Resource that is represented by the Managed Object is not otherwise visible to management. This concept of a Managed Object boundary has no implications for implementation, but provides an architectural distinction between the definitions to be developed by Managed Objects definers (for example, layer groups), which are at the boundary, and the definitions and specifications of the remainder of systems management, which are at and outside the boundary. The object model specifies that generic objects are defined, from which specific Managed Object instances can be instantiated. In general, object instances are created or deleted:

- by management protocol interactions
- as a result of the operation of the Resource to which they relate by other means, for example, when a disk drive goes on-line.

Managed object classes are arranged in a hierarchy according to commonality of state and operation. Thus the object model provides an organised presentation of systems management.

The design of systems management requires an approach to be adopted that will allow the specification to be standardised in a modular fashion and provide for extensibility of the protocol and procedures. The information model makes use of object-oriented design principles because they provide the above capabilities and provide for a well managed reuse of previously defined specifications. It should be noted that whereas the model is described using object-oriented techniques, such techniques need not be used in implementing systems management.

The X/Open Systems Management Model must encompass Managed Object classes to be used with CMIP, Management Information Bases (MIBs) to be used with SNMP, and OMG-based management technology. In a mixed environment, such as X/Open Systems Management, there is the need for management protocols and sets of management information to coexist. The following sections describe the definitions of Managed Objects and MIBs and discuss how mapping may be performed between the various models.

The OSI Management Information Model, Internet Information Model, and the implicit CORBA object model, have each developed independently of one another. There is the expectation that further object or information models will be required to satisfy the needs of other technologies — for example, Databases, Directory Services, etc. We risk proliferation of object models and of mappings between object models.

To avoid the divergences all this implies, the OMG is working on developing an Object Model that will present a common *core* object model, yet will allow technology-specific extensions — which they call *profiles* — to be defined.

A group of experts are cooperating to develop a mapping between the OSI Management Information Model, and later the Internet Information Model, and the OMG Object Model. It is hoped that this may be achieved by creating profiles that extend the OMG core object model for OSI management and Internet management. This approach will contribute towards interoperability of management applications based on these coexisting object models.

Reduction in number of object models will contribute towards interoperability of management applications.

4.2 OMG Object Model

This section describes the concrete object model which underlies the Common ORB Architecture (see reference **CORBA**). The model is derived from the abstract object model defined by the Object Management Group (reference **OMAG**).

4.2.1 Overview

The object model provides an organised presentation of object concepts and terminology. It defines a partial model for computation that embodies the key characteristics of objects as realised by the submitted technologies.

The OMG object model is abstract in that it is not directly realised by any particular technology. The model described here is a concrete object model. A concrete object model may differ from the abstract object model in several ways:

- It may *elaborate* the abstract object model by making it more specific, for example, by defining the form of request parameters or the language used to specify types.
- It may *populate* the model by introducing specific instances of entities defined by the model, for example, specific objects, specific operations, or specific types.
- It may *restrict* the model by eliminating entities or placing additional restrictions on their use.

An object system is a collection of objects that isolates the requestors of services (clients) from the providers of services by a well-defined encapsulating interface. In particular, clients are isolated from the implementations of services as data representations and executable code.

The object model first describes concepts that are meaningful to clients, including such concepts as object creation and identity, requests and operations, types and signatures. It then describes concepts related to object implementations, including such concepts as methods, execution engines and activation.

The object model is most specific and prescriptive in defining concepts meaningful to clients. The discussion of object implementation is more suggestive, with the intent of allowing maximal freedom for different object technologies to provide different ways of implementing objects. See **CORBA** Chapter 9 for more information on implementation rules for objects which are managed by the Basic Object Adapter.

There are some other characteristics of object systems that are outside the scope of the object model. Some of these concepts are aspects of application architecture, and some are associated with specific domains to which object technology is applied. Such concepts are more properly dealt with in an architectural reference model. Examples of excluded concepts are compound objects, links, copying of objects, change management and transactions. Also outside the scope of the object model is the model of control and execution.

This object model is an example of a classical object model, where a client sends a message to an object. Conceptually, the object interprets the message to decide what service to perform. In the classical model, a message identifies an object and zero or more actual parameters. As in most classical object models, a distinguished first parameter is required, which identifies the operation to be performed; the interpretation of the message by the object involves selecting a method based on the specified operation. Operationally, of course, method selection could be performed either by the object or the ORB.

4.2.2 Object Semantics

An object system provides services to *clients*. A client of a service is any entity capable of requesting the service. This section defines the concepts associated with object semantics, that is, the concepts relevant to clients.

4.2.2.1 Objects

An object system includes entities known as *objects*. An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.

4.2.2.2 Requests

Clients request services by issuing *requests*. A request is an event, that is, something that occurs at a particular time. The information associated with a request consists of an operation, a target object, zero or more (actual) parameters, and an optional request context.

A *request form* is a description or pattern that can be evaluated or performed multiple times to cause the issuing of requests. As described in **CORBA** Chapter 4 request forms are defined by particular language bindings. An alternative request form consists of calls to the dynamic invocation interface to create an invocation structure, to add arguments to the invocation structure, and to issue the invocation.[1]

A *value* is anything that may be a legitimate (actual) parameter in a request. A value may identify an object, for the purpose of performing the request. A value that identifies an object is called an object name. More particularly, a value is an instance of an IDL (Interface Definition Language, defined in **CORBA** Chapter 4) data type.

An *object reference* is an object name that reliably denotes a particular object. Specifically, an object reference will identify the same object each time the reference is used in a request (subject to certain pragmatic limits of space and time). An object may be denoted by multiple, distinct object references.

A request may have parameters that are used to pass data to the target object; it may also have a request context which provides additional information about the request.

A request causes a service to be performed on behalf of the client. One outcome of performing a service is returning to the client the results, if any, defined for the request.

If an abnormal condition occurs during the performance of a request, an exception is returned. The exception may carry additional return parameters particular to that exception.

The request parameters are identified by position. A parameter may be an input parameter, an output parameter, or an input-output parameter. A request may also return a single *result value*, as well as any output parameters.

The following semantics hold for all requests:

- Any aliasing of parameter values is neither guaranteed removed nor guaranteed preserved.
- The order in which aliased output parameters are written is not guaranteed.
- Any output parameters are undefined if an exception is returned.

1. Descriptions of these requests are in **CORBA** Chapter 5 for the C-language binding for IDL and Chapter 6 for the dynamic invocation interface.

- The values which may be returned in an input-output parameter may be constrained by the value which was input.

Descriptions of the permitted values in requests and the permitted exceptions may be found in Section 4.2.2.4 and **Exceptions** on page 19.

4.2.2.3 Object Creation and Destruction

Objects can be created and destroyed. From a client's point of view, there is no special mechanism for creating or destroying an object. Objects are created and destroyed as an outcome of issuing requests. The outcome of object creation is revealed to the client in the form of an object reference that denotes the new object.

4.2.2.4 Types

A *type* is an identifiable entity with an associated predicate (a single-argument mathematical function with a boolean result) defined over values. A value satisfies a type if the predicate is true for that value. A value that satisfies a type is called a member of the type.

Types are used in signatures to restrict a possible parameter or to characterise a possible result.

The extension of a type is the set of values that satisfy the type at any particular time.

An object type is a type whose members are objects (literally, values that identify objects). In other words, an object type is satisfied only by (values that identify) objects.

Data types in this model are constrained as follows:

- Basic types:
  - — 16-bit and 32-bit signed and unsigned 2's complement integers
  - — 32- bit and 64-bit IEEE floating point numbers
  - — characters, as defined in the ISO 8859-1:1987 standard
  - — a boolean type taking the values TRUE and FALSE
  - — an 8-bit opaque data type, guaranteed to not undergo any conversion during transfer between systems
  - — enumerated types consisting of ordered sequences of identifiers
  - — a string type which consists of a variable-length array of characters; the length of the string is available at run time
  - — a type **any** which can represent any possible basic or constructed type.
- Constructed types:
  - — a record type (called struct), consisting of an ordered set of (name,value) pairs
  - — a discriminated union type, consisting of a discriminator followed by an instance of a type appropriate to the discriminator value
  - — a sequence type which consists of a variable-length array of a single type; the length of the sequence is available at run time
  - — an array type which consists of a fixed-length array of a single type
  - — an interface type, which specifies the set of operations which an instance of that type must support.

Values in a request are constrained to values which satisfy these type constraints. The legal values are shown in Figure 4-1. No particular representation for values is defined.

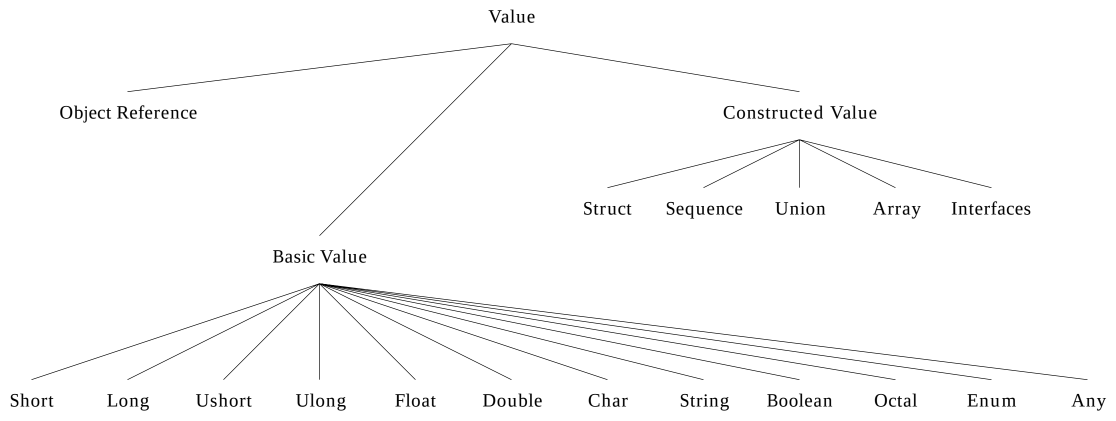

Figure 4-1 Legal Values

4.2.2.5 *Interfaces*

An *interface* is a description of a set of possible operations that a client may request of an object. An object satisfies an interface if it can be specified as the target object in each potential request described by the interface.

An interface type is a type that is satisfied by any object (literally, any value that identifies an object) that satisfies a particular interface.

Interfaces are specified in IDL. Interface inheritance provides the composition mechanism for permitting an object to support multiple interfaces. The principal interface is simply the most-specific interface that the object supports, and consists of all operations in the transitive closure of the interface inheritance graph.

4.2.2.6 *Operations*

An *operation* is an identifiable entity that denotes a service that can be requested.

An operation is identified by an operation identifier. An operation is not a value.

An operation has a *signature* that describes the legitimate values of request parameters and returned results. In particular, a signature consists of:

- a specification of the parameters required in requests for that operation
- a specification of the result of the operation
- a specification of the exceptions that may be raised by a request for the operation and the types of the parameters accompanying them
- a specification of additional contextual information that may affect the request
- an indication of the execution semantics the client should expect from a request for the operation.

Operations are (potentially) generic, meaning that a single operation can be uniformly requested on objects with different implementations, possibly resulting in observably different behaviour. Genericity is achieved in this model via interface inheritance in IDL and the total decoupling of implementation from interface specification.

The general form for an operation signature is:

```
[oneway]<op_type_spec> <identifier> (param1, . . ., paramL)
   [raises(except1,. . .,exceptN)] [context(name1, . . ., nameM)]
```

where:

- The optional **oneway** keyword indicates that best-effort semantics are expected of requests for this operation; the default semantics are exactly-once if the operation successfully returns results or at-most-once if an exception is returned.
- The **<op_type_spec>** is the type of the return result.
- The **<identifier>** provides a name for the operation in the interface.
- The operation parameters needed for the operation; they are flagged with the modifiers **in**, **out** or **inout** to indicate the direction in which the information flows (with respect to the object performing the request).
- The optional **raises** expression indicates which user-defined exceptions can be signalled to terminate a request for this operation; if such an expression is not provided, no user-defined exceptions will be signalled.
- The optional **context** expression indicates which request context information will be available to the object implementation; no other contextual information is required to be transported with the request.

Parameters

A parameter is characterised by its mode and its type. The mode indicates whether the value should be passed from client to server (**in**), from server to client (**out**), or both (**inout**). The parameter's type constrains the possible value which may be passed in the direction or directions dictated by the mode.

Return Result

The return result is a distinguished **out** parameter.

Exceptions

An exception is an indication that an operation request was not performed successfully. An exception may be accompanied by additional, exception-specific information.

The additional, exception-specific information is a specialised form of record. As a record, it may consist of any of the types described in Section 4.2.2.4 on page 17.

All signatures implicitly include the standard exceptions described in **CORBA** Chapter 4.

Contexts

A request context provides additional, operation-specific information that may affect the performance of a request.

Execution Semantics

Two styles of execution semantics are defined by the object model:

at-most-once	If an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at-most-once.
best-effort	A best-effort operation is a request-only operation, that is, it cannot return any results and the requester never synchronises with the completion, if any, of the request.

The execution semantics to be expected are associated with an operation. This prevents a client and object implementation from assuming different execution semantics.

Note: A client is able to invoke an at-most-once operation in a synchronous or deferred-synchronous manner.

4.2.2.7 Attributes

An interface may have attributes. An attribute is logically equivalent to declaring a pair of accessor functions: one to retrieve the value of the attribute and one to set the value of the attribute.

An attribute may be read-only, in which case only the retrieval accessor function is defined.

4.2.3 Object Implementation

This section defines the concepts associated with object implementation, that is, the concepts relevant to realising the behaviour of objects in a computational system.

The implementation of an object system carries out the computational activities needed to effect the behaviour of requested services. These activities may include computing the result of the request and updating the system state. In the process, additional requests may be issued.

The implementation model consists of two parts: the execution model and the construction model. The execution model describes how services are performed. The construction model describes how services are defined.

4.2.3.1 The Execution Model: Performing Services

A requested service is performed in a computational system by executing code that operates upon some data. The data represents a component of the state of the computational system. The code performs the requested service, which may change the state of the system.

Code that is executed to perform a service is called a *method*. A method is an immutable description of a computation that can be interpreted by an *execution engine*. A method has an immutable attribute called a *method format* that defines the set of execution engines that can interpret the method. An execution engine is an abstract machine (not a program) that can interpret methods of certain formats, causing the described computations to be performed. An execution engine defines a dynamic context for the execution of a method. The execution of a method is called a *method activation*.

When a client issues a request, a method of the target object is called. The input parameters passed by the requestor are passed to the method, and the output parameters and return value (or exception and its parameters) are passed back to the requestor.

Performing a requested service causes a method to execute that may operate upon an object's persistent state. If the persistent form of the method or state is not accessible to the execution engine, it may be necessary to first copy the method or state into an execution context. This process is called activation; the reverse process is called deactivation.

4.2.3.2 *The Construction Model*

A computational object system must provide mechanisms for realising behaviour of requests. These mechanisms include definitions of object state, definitions of methods, and definitions of how the object infrastructure is to select the methods to execute and to select the relevant portions of object state to be made accessible to the methods. Mechanisms must also be provided to describe the concrete actions associated with object creation, such as association of the new object with appropriate methods.

An *object implementation* — or *implementation*, for short — is a definition that provides the information needed to create an object and to allow the object to participate in providing an appropriate set of services. An implementation typically includes, among other things, definitions of the methods that operate upon the state of an object. It also typically includes information about the intended type of the object.

4.3 ISO Management Model

4.3.1 Objects

4.3.1.1 Introduction

Important base documents to the ISO System Management Model are:

- ISO/IEC 10165-1 (CCITT X.720) Structure of Management Information - Part 1: Management Information Model (reference **MIM**)
- ISO/IEC 10165-4 (CCITT X.722) Structure of Management Information - Part 4: Guideline for the Definition of Managed Objects (reference **GDMO**).

ISO/IEC JTC1 SC21/WG4 and CCITT SGVII are jointly responsible for the development of International Standards and Recommendations that describe the architecture for OSI management, the services, protocols and functions that are used for systems management and the structure of management information. The MIM and GDMO are primarily aimed at communications management, but the principles involved are equally applicable to systems managements and hence are of interest to X/Open Systems Management.

The MIM and GDMO specifications provide developers of Managed Object class definitions with the information and documentation tools required in order to produce consistent Managed Object class definitions that are compatible with OSI management standards developed jointly by ISO/IEC and CCITT. The MIM defines the general means by which Managed Objects are created and deleted, and how they are assigned initial values. The GDMO addresses some global issues (such as registration), gives advice on a number of areas of Managed Object behaviour, and then proceeds to describe notational tools for Managed Object definition. In Section 4.3.4 on page 31 a number of templates for Managed Object definition are described.

The guidelines and notational tools defined in GDMO have wide support, and are being used not only for communications management but also for systems management.

The notational tools described in GDMO are the templates that may be used to define Managed Objects. Again, these are applicable to both communications management and systems management.

A Managed Object is an abstract view of a Resource in the systems management domain. The object-oriented model used in systems management allows objects of a similar type to be grouped together to form what are known as classes. Objects are of a similar type if they have the same data structures and exhibit the same functionality. A Managed Object is thus an instance of a Managed Object class.

The object-oriented approach makes it possible to attach to a Managed Object functionalities that are specifically permitted for it. The object is thus ‘‘encapsulated’’ and can only be addressed via specified operations that are included in the object definition. Thus the object-oriented approach abstracts the features of objects enabling *classes* to be defined that contain all objects of similar structure. This obviates the need to redescribe structure for every single object class. It is important to note that structure includes not only the object’s data but also the operations that may be performed on the object’s data.

ISO defines a Managed Object class as the set of attributes, operations, notifications and behaviour definitions to which a Managed Object class name has been allocated. Managed object class definitions are documented by means of a Managed Object class template and one or more other templates which are directly or indirectly referenced by the Managed Object class templates. This section defines attributes, operations, notifications and behaviour (subsequent sections define structuring techniques and templates).

In outline the structure of an object class can be represented as follows:

```
<object class c2>
  derived from <object class c1> ...
  supports mandatory packages <pm1> <pm2> ...
  supports conditional packages <pc1> <pc2> ...
```

In turn, the structure of a package can be summarised as:

```
<package p1>
  possesses attributes <a1>, <a2> ...
  supports actions <o1>, <o2> ...
  emits notifications <n1>, <n2> ...
  supports behaviours <b1> <b2> ....
```

Attributes, actions and notifications can be defined independently of packages and Managed Objects. Each definition can be allocated its own unique OBJECT IDENTIFIER and can thus be used in more than one Managed Object definition. (An OBJECT IDENTIFIER is an administratively assigned, globally unique label.)

4.3.1.2 Attributes

Managed objects have attributes. The value of an attribute can determine or reflect the behaviour of a Managed Object. The operations that can be performed on a particular attribute are specified in the definition of the Managed Object.

An attribute can have internal structure, that is, it can consist of a set or a sequence of values. In general, attributes can be read or modified by sending a request to the Managed Object to read (get) or write (replace) the value. Additional operations are defined for set-valued attributes.

The Managed Object definition may optionally include initial and default values for attributes. Furthermore, value restrictions on an attribute can be specified in terms of a permitted value set and a required value set. The permitted value set specifies all values that the attribute is permitted to take; this is a subset of the values of the syntax of the attribute. The required value set is a subset of the permitted values and defines all the values that attribute must support (the required value set may be empty). However, particular assignments may be subject to other constraints, for example, access control. Value restrictions are illustrated in the example in Figure 4-2.

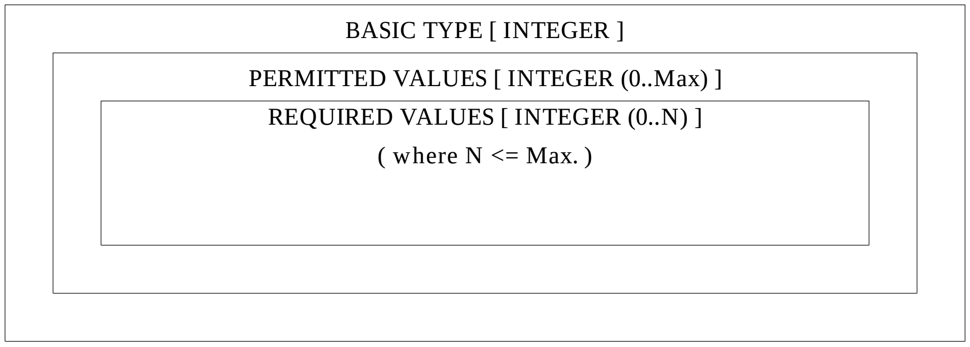
BASIC TYPE [INTEGER]
PERMITTED VALUES [INTEGER (0..Max)]
REQUIRED VALUES [INTEGER (0..N)]
(where N <= Max.)

Figure 4-2 Attribute Value Restrictions

4.3.1.3 Operations

The ISO Management Information Model defines two types of operations:

- those that can be applied to the Managed Object as a whole
- those that are sent to the Managed Object in order to be applied to its attributes.

Operations that may be performed on a Managed Object are:

- *Create Managed Object*
  This operation instantiates a Managed Object class. The model allows Managed Objects to be created by other than management action, therefore, not all Managed Object classes need support the create operation. Templates associated with the Managed Object class include the precise semantics of creation.
- *Delete Managed Object*
  This operation requests the Managed Object to delete itself. Templates associated with the Managed Object class include the precise semantics of deletion.
- *Action*
  This operation requests the Managed Object to perform a specified action and optionally to indicate the result of that action.

Operations on attributes are defined to be performed upon the Managed Object that contains the attribute and not directly on the attribute. The Managed Object is therefore able to enforce constraints on attribute values, including constraints between the values of individual attributes. The following management operations can be applied to attributes:

- get attribute value
- replace attribute value
- replace with default value
- add member (applies to set-valued attributes)
- remove member (applies to set-valued attributes).

The Managed Object definitions may contain consistency constraints that must be satisfied in order for the operation to be performed (this includes relationships to be maintained between values of different attributes). In addition, there may be access control restrictions to be observed.

Where a management system is requested to perform operations across several Managed Objects, it may do so with atomic synchronisation (that is, either all operations succeed or none are performed) or on a best effort basis.

4.3.1.4 Attribute Groups

An attribute group allows a collection of attributes to be identified by a single name. Attribute Groups can be identified in management operations; for example CMIS operations, like M-Get and M-Set, may refer to Attribute Groups. There are two types of attribute groups:

Fixed	This is an attribute group whose members are defined as part of the initial attribute group definition and whose members may not subsequently be expanded.
Extensible	This is an attribute group in which attributes may be added as members as a result of specialisation. The attributes specified for each extension may be defined in the same conditional package as the attribute or in a mandatory package.

4.3.1.5 Notifications

Managed objects may emit notifications when events occur. Notifications are specific to the Managed Object that emit them, and the information they contain is part of the definition of the Managed Object class of which the Managed Object is an instance. Notifications can be defined independently of Managed Object definitions and can then be referenced by more than one Managed Object class definition. However, the meaning of a notification must be considered in the context of the class of the Managed Object that emitted it.

4.3.1.6 Behaviour

A Managed Object class includes the definition of the object class' behaviour. Behaviour is expressed in natural language and describes the following:

- the semantics of the object's attributes, operations and notifications
- the response to management operations that may be invoked on the Managed Object
- the circumstances under which notifications may be emitted
- the dependencies between values of particular attributes
- the effects of relationships between different Managed Objects
- consistency constraints on attributes
- preconditions that identify the conditions when operations and notifications can be assumed to have valid meaning
- postconditions that identify the results of the processing of a management operation or the emission of a notification
- invariants that are in effect for the entire lifetime of the Managed Object and that describe conditions that are true for operations of the Managed Object
- synchronisation properties of the Managed Object.

4.3.1.7 Parameters

The GDMO Parameter template is a notational tool that can be used to specify syntax and semantics of an open-ended ASN.1 type (for example, ANY DEFINED BY) appearing in other GDMO templates, such as ATTRIBUTE, PACKAGE, ACTION and NOTIFICATION. The parameter template is best explained using an example.

The following information syntax is associated with an ISO ‘‘communicationsAlarm’’ notification:

```
AlarmInformation ::= SEQUENCE {
     severity                  Severity,
     additionalInformation     SET OF ManagementExtension OPTIONAL}

ManagementExtension ::= SEQUENCE {
     identifier         OBJECT IDENTIFIER,
     criticality        BOOLEAN,
     data               ANY DEFINED BY identifier}
```

At the time of defining the ‘‘communicationsAlarm’’ notification, the syntax for different management extensions are not known. In other words, an open-ended type is defined so that the *hole* can be filled later.

In this example, GDMO parameter templates may be used to register the syntax of a ‘‘ManagementExtension’’. A parameter label associates the value of an object identifier with a specific ASN.1 syntax for the parameter ‘‘data’’ in ‘‘ManagementExtension’’. One or more parameter labels can then be used to augment a package definition containing the ‘‘communicationsAlarm’’ notification. When a ‘‘communicationsAlarm’’ is emitted by an object with the augmented package, the notification may contain the syntax registered by the parameter label in the ‘‘additionalInformation’’ parameter. In addition to the syntax, there may be semantics or behaviour associated with the registered parameter.

The parameter template may also be used to specify the syntax and possibly the behaviour associated with specific errors for the different GDMO templates (such as ATTRIBUTE or ACTION). Each parameter template must specify a context in which the registered syntax can be used (for example, EVENT-INFO or SPECIFIC-ERROR).

4.3.1.8 Syntax

The definitions of Managed Object classes allow the specification of the syntax to be used with components of the Managed Object, such as attributes and notifications. The syntax specification identifies the ASN.1 data type that describes the structure of the information that is carried in the management protocol.

4.3.2 Structure/Relationships

4.3.2.1 Introduction

The Managed Object model supports a number of techniques for representing the structure in Managed Objects and to reflect the groupings of data or functionality. Structuring techniques can be static or dynamic. Static structuring techniques are used to represent common structure that is present in the object class definition; examples are mandatory packages and attribute groups. Dynamic structuring techniques are used to define common structure that is present at implementation, installation or instantiation time; examples are conditional packages and containment.

The remainder of this section describes structuring techniques used in the ISO systems management object model.

4.3.2.2 Specialisation/Inheritance

Instances of Managed Objects that share the same operations, attributes, packages, notifications and behaviour are said to be of the same Managed Object class. Specialisation is the process by which a new Managed Object class can be derived from one or more other Managed Object classes. A Managed Object class that is a specialisation of another class is known as a subclass of that class (its superclass). The subclass inherits the operations, attributes, notifications, packages and behaviour of its superclass, and differs from it by adding:

- new management operations
- new attributes
- new notifications
- new behaviour
- extensions to the characteristics of its superclass(es).

The hierarchical arrangement of Managed Objects created by the process of specialisation is called a class hierarchy. A subclass is a Managed Object class that is a specialisation of another Managed Object class and is lower in the hierarchy than the class of which it is a specialisation. One Managed Object class, called **top**, is designated the ultimate superclass in the Managed Object class hierarchy; **top** cannot be instantiated.

The conceptual mechanism by which attributes, operations and behaviour are acquired by a subclass from its superclass is called **Inheritance**. Systems management supports only **Strict Inheritance**; this allows specialisation only by the *addition* of new attributes, operations, notifications or behaviour but not by the deletion of any of the parent class characteristics. Systems management also allows **Multiple Inheritance**. This mechanism allows a subclass to acquire attributes, notifications, operations and behaviour from more than one superclass.

4.3.2.3 *Uninstantiable Managed Object Classes*

Some Managed Object classes cannot be instantiated. They are used to provide a common base from which to specialise classes, thus ensuring that two or more Managed Objects classes have a common set of characteristics. For example, a generic virtual circuit Managed Object class may be defined, which can have permanent and switched virtual circuit Managed Object classes as subclasses.

Uninstantiable Managed Object classes may also be used in the definition of relationships. For example, a superclass might be defined with the relationship properties which should be inherited by subclasses which have the specified relationship. Such uninstantiable Managed Object classes are also known as abstract classes.

4.3.2.4 *Packages*

A package is a collection of characteristics of Managed Objects which is treated as a single module in the definition of a Managed Object class. The characteristics included in a package definition consist of attributes, notifications, operations and behaviour. Packages may be *mandatory* or *conditional*. All components of a mandatory package are present in all instances of a given Managed Object. A conditional package has an explicit condition associated with it, which determines if the conditional package is present when the Managed Object class is instantiated.

Conditions that determine the absence or presence of conditional packages are related to the capability of the underlying Resource being modelled or to the presence/absence of management functions in the managed system. For example, in a Managed Object covering OSI transport protocol, conditional packages will model optional protocol functions. Conditional packages allow small variations in Managed Objects to be handled without the need to specialise new Managed Object classes, thus limiting the number of Managed Object classes and object identifiers.

The presence of packages in a Managed Object can be ascertained. All Managed Objects inherit from **top** an attribute that is a set of object identifiers corresponding to those instantiated packages that are registered. With the exception of the Packages attribute, packages are not visible at the Managed Object boundary; specifically they cannot be used in Management Services operations nor can they be used as part of scoping rules.

4.3.2.5 *Interoperability*

The use of options and conditional packages is intended to limit the number of different Managed Object classes. However, this means that a given Managed Object may represent more than one Resource and hence may behave differently in different instances. The potential problem of interoperability is overcome either at the agent system or at the managing system.

- When *interoperability* for a given Managed Object is provided by the agent systems implementing the object, then *allomorphism* (described below) is used. When an allomorphic object is created, a number of compatibility rules (described in GDMO) must be adhered to.

- When *interoperability* for a given Managed Object is provided by managing systems, the Managed Object always responds according to its actual class definition. The managing system is required to handle any additional information that it does not understand or expect (for example, by ignoring this information). This method does not impose any compatibility restrictions when new Managed Objects are created by specialisation, but adherence to the compatibility rules required for allomorphism makes the Management Task easier.

4.3.2.6 Allomorphism

Allomorphism is the ability of a Managed Object of a given class to resemble the behaviour of another class as observed by systems management protocols. It allows different views of Resources to be presented to different managing systems and to hide optional facilities from some managing systems. It thus allows a Managed Object class definition to be extended in a way that permits interoperability when either the Manager or the Managed Object does not include the relevant extension.

Allomorphism is intended to allow, to the extent possible, a managing system to manage a system of high capability as if it was a system of lesser capability. Specifically, it is a requirement that if none of the extended capabilities are required, management must be possible as effectively as if the managed system did not have them.

An example where allomorphism may be used would be to extend a Managed Object class to reflect:

- extensions to a standardised view of a Resource; for example, where a Managed Object supporting 1988 X.25 is an extension of one which supports 1984 X.25
- vendor-specific extensions to a standardised view of Resources; for example, where a Managed Object supporting vendor X's implementation of X.25 is an extension of the standard view of 1988 X.25.

Allomorphism cannot be used to allow a management agent system to provide a restricted view of a Managed Object to a managing system, because a management agent system cannot enforce a particular allomorphic Managed Object definition. If it receives a valid request for an operation, then (subject to access control restrictions) it must perform it.

An allomorphic class of a given Managed Object may be one of the superclasses of that Managed Object's class in the inheritance hierarchy. However, this is not a requirement for allomorphism.

4.3.2.7 Containment

Containment is a structuring relationship between Managed Object instances (not classes). Containment allows a Managed Object of one class to contain other Managed Objects of the same or different classes. The contained Managed Object is said to be the subordinate Managed Object and the containing Managed Object the superior Managed Object. The existence of the subordinate Managed Object is dependent on the existence of the superior Managed Object. A Managed Object can be contained within one and only one containing Managed Object.

The specification of a containment relationship can be used to model real-world hierarchies; these may be physical (for example, an instance of a Managed Object describing a port may have a containment relationship to an instance of a Managed Object describing a router) or logical (for example, an instance of a Managed Object describing a user may have a containment relationship to an instance of a Managed Object describing a particular group of users - for example, students).

4.3.2.8 Filters

Filters allow a management function to be applied to a number of Managed Objects that are selected because their attributes meet specified criteria. Filtering is explicitly supported by the CMIS services (M-Set, M-Get, M-Delete and M-Action). The managing system can specify that operations to be performed on Managed Objects selected by filters are to be atomic (that is, they either all succeed or all fail) or on a best effort basis. Filtering can only be usefully performed on primitive types. Filtering can be applied together with scoping, in which case scoping is always applied first.

A filter is an assertion about the presence or the value of an attribute in a Managed Object, and is satisfied if, and only if, it evaluates to TRUE. A filter may be composed of a number of simpler filters using logical operators *and*, *or* and *not*; this is referred to as a nesting. An assertion about the value of an attribute is evaluated only if the attribute is present in the Managed Object. If the attribute is not present, the attribute assertion for that attribute is assigned the value FALSE.

Filtering on attributes that are not primitive types requires that all values of the attribute be specified. In other words, if the attribute Person has fields age and haircolour, one cannot filter Person.age=35 but Person.age=35 and Person.haircolour=brown. Furthermore, it is not always possible to define attribute assertions that always evaluate to TRUE. Therefore, it is generally preferable to define an attribute to be used in filtering as a primitive attribute.

Matching rules can be defined for each attribute, restricting or identifying types of filters that can be applied to the attribute (for example, MATCHES FOR ORDERING, SUBSTRINGS).

4.3.2.9 Scoping

Scoping allows a subset of the objects to be selected for application of a specified operation by specifying a starting point in the Managed Object containment hierarchy and a depth. The CMIS M-Set, M-Get, M-Delete and M-Action services allow a scoping parameter that selects:

- a base object alone
- the *n*th level subordinate of a base object
- a base object and all its subordinates down to and including the *n*th level
- a base object and all its subordinates.

4.3.2.10 Relationships

General relationships between Managed Objects are possible. They exist when the properties of one Managed Object affect the properties of other (related) Managed Object(s). They are defined by their behaviour and may be subtyped on this basis. For example, changing the attribute of a (target) Managed Object may result in a change in the behaviour or the modification of another attribute both in the target Managed Object and in another related Managed Object. The relationship between Managed Objects is described in the ISO draft General Relationship Model (see reference **GRM**). Containment, as described previously, is one example of a relationship.

4.3.2.11 Attributes of top

Each Managed Object contains all the information needed to describe itself for the purposes of management access. Each Managed Object class is derived directly or indirectly from **top**, and it inherits from **top** the following attributes:

Managed Object Class	This attribute identifies the class of the Managed Object.

Allomorphs	This attribute identifies the set of allomorphic classes of this Managed Object. It is in a conditional package and is present only if the Managed Object instance supports allomorphic class.
Name Binding	This attribute identifies the object identifier of the binding that is in use between the Managed Object and its superior.
Packages	This attribute identifies the packages that have been instantiated. Its value is a set of object identifiers corresponding to those instantiated packages that are registered. This attribute is in a conditional package that is instantiated only if it would not be empty.

4.3.3 General Principals for Managed Object Definition

GDMO gives guidance on a number of issues:

- Managed object class definers should strive to identify and use definitions that appear in standards or that have been produced by other groups in order to increase commonality.
- Grouping of data or functionality of Managed Objects should be represented through the use of a number of structuring techniques (described in Section 4.3.2 on page 26. These include:
  - — attribute groups
  - — subclasses (specialisation)
  - — multiple inheritance
  - — containment
  - — packages
  - — relationships.
- In order to ensure compatibility of Managed Object classes to their superclasses, the rules for inheritance limit:
  - — the ways in which the required and permitted value sets of attributes of a Managed Object class can be modified
  - — the ability to add parameters to actions and notifications.

  For this reason, it is necessary to try to anticipate future uses of Managed Object classes and define them with suitable extension capabilities. This may include the definition of unrestricted Managed Object classes (which will not be instantiated) from which restricted Managed Object classes may be defined.
- A value set for an attribute may be defined by:
  - — defining the attribute value statically, as part of the definition of the Managed Object class
  - — defining a second attribute, whose value indicates the value set the attribute may contain.

  The former method minimises the number of attribute definitions of a Managed Object class, while the latter minimises the number of Managed Object subclasses to handle different value set variants.
- Attribute groups should be used in preference to complex attribute types where it is required to individually modify elements of the attribute.
- The effects of concurrent update on attributes that can be updated by normal operation and management action should be defined if possible.

- Managed objects that can be instantiated must include at least one value that can be used for naming. Normally, when a Managed Object is deleted the value of the attribute used for naming may be reused, however, for some objects an additional naming attribute whose value is maintained unique over time may also be defined.
- Where appropriate, the standard Create object, Delete object, Replace attribute value, Replace attribute with default, Add / Remove Member or Get attribute operations should be used; in all other circumstances the Action procedure should be used. Examples where the Action operation should be used include the following:

— where the other operations, together with scoping and filtering, cannot define the required action

— where operations on more than one Managed Object are required as an atomic operation.

4.3.4 Templates

GDMO defines templates that can be used to define Managed Objects. A template is a high-level, structured description of a Managed Object class; it excludes syntactic detail and can be viewed as a form to be filled in by a designer. The use of templates is intended to make the work of defining Managed Objects more consistent and more structured.

A Managed Object template identifies the inheritance relationships that exist between the Managed Object class and other Managed Object classes, and the packages of behaviour, attributes, notifications and operations that are included in the Managed Object class definition (which have been discussed in Section 4.3.2 on page 26 and Section 4.3.3 on page 30). Details of the Managed Object's attributes, behaviour, actions, notifications, parameters and packages can be specified through a number of subsidiary templates.

A template consists of:

- a specification label
- a template type
- a lists of keywords
- an ASN.1 object identifier.

The specification label is an ASN.1 value reference (ASN.1 — Abstract Notation Number 1 — is an internationally standardised formal language for describing structured information). It is allocated to a completed template and must be distinguishable from all other such values. It is used to refer to the completed template (that is, the Managed Object specification) in other (human-readable) documents, while the ASN.1 object identifier is used to identify the specification when a managing system (a Manager) interacts with a managed system (an Agent). Stand-alone systems should also use ASN.1 object identifiers to ensure globally unique identifiers, thus allowing future networking expansion. The label has no semantic significance in itself and any label could be substituted for another without changing the meaning, provided the new label is used consistently and does not clash with any other labels already in use.

Templates for MANAGED OBJECT CLASS, PACKAGE, PARAMETER, NAME BINDING, ATTRIBUTE, ATTRIBUTE GROUP, BEHAVIOUR, ACTION and NOTIFICATION are defined.

4.3.5 IS GDMO Template

4.3.5.1 Managed Object Class Template

GDMO defines a MANAGED OBJECT CLASS template to be used for the top-level definition of a Managed Object class. A number of supporting templates define elements of the Managed Object class template (for example, the attribute template). The MANAGED OBJECT CLASS template is listed below and its elements described. The following notational conventions are used:

- the square brackets, [and], enclose clauses that are allowed to be empty in a completed template. An empty clause may be omitted (in which case the entire clause including the keyword which introduces it should be omitted). If the square brackets are followed by a *, then the enclosed clause may be repeated zero or more times.
- The braces { and } indicate a list.

The IS Managed Object class template is, of course, derived from the DIS Managed Object class template. The main difference between the two templates is the use of packages in the former to group and indicate as optional or mandatory the elements of the template (the DIS version indicated mandatory/optional elements by means of MAY CONTAIN and MUST CONTAIN clauses). The DIS Managed Object template is not used any more and X/Open only endorses the IS template. The DIS template has been used in the past and users may encounter Managed Object definitions that use it; however, groups that have used it have now migrated to the IS version.

The IS GDMO Managed Object class template is shown below.

```
<class-label> MANAGED OBJECT CLASS
  [DERIVED FROM          <class-label>   [,<class-label>]* ;
  ]
  [CHARACTERIZED BY        <package-label>  [,<package-label>]*;
  ]
  [CONDITIONAL PACKAGES   <package-label>  PRESENT IF condition-definition
                        [,<package-label>  PRESENT IF condition-definition]*;
  ]
REGISTERED AS object-identifier;
```

- DERIVED FROM
  The DERIVED FROM clause specifies the Managed Object classes (superclasses) from which the Managed Object being defined has been specialised. As multiple inheritance is allowed, a Managed Object class may have more than one superclass. The DERIVED FROM clause causes the Managed Object class to inherit all characteristics of its superclass(es), including behaviour. These may then be augmented or modified by the CHARACTERISED BY and CONDITIONAL PACKAGES clauses. The DERIVED FROM clause is mandatory in all Managed Objects except **top**.
- CHARACTERIZED BY
  The CHARACTERIZED BY clause is optional, and if present allows one or more mandatory packages to be specified. A package that is conditional in the object class' superclass may be referenced in the CHARACTERIZED BY clause in order to make it mandatory.
- CONDITIONAL PACKAGES
  The CONDITIONAL PACKAGES clause is optional and, if present, allows one or more packages to be specified. The PRESENT IF construct allows the condition(s) that must be met for the package to be included in an instance of the class. The *condition-definition* is expressed in natural language.

- REGISTERED AS
  The REGISTERED AS clause allows a globally unique identifier for the object class to be defined.

4.3.5.2 *Package Template*

The PACKAGE template includes clauses to define the behaviour, attributes, attribute groups actions and notifications found in the GDMO template. Mandatory and conditional packages have the same format, that is, the same template can be used to define both. The PACKAGE template is shown below:

```
<package-label> PACKAGE
  [BEHAVIOUR           <behaviour-definition-label>
                      [,<behaviour-definition-label>]*;
  ]
  [ATTRIBUTES          <attribute-label> propertylist [<parameter-label>]*
                      [,<attribute-label> propertylist [<parameter-label>]*]*;
  ]
  [ATTRIBUTE GROUPS    <group-label> [<attribute-label>]*
                      [,<group-label> [<attribute-label>]*]*;
  ]
  [ACTIONS             <action-label> [<parameter-label]*
                      [,<action-label> [<parameter-label]*]*;
  ]
  [NOTIFICATIONS       <notification-label> [<parameter-label>]*
                      [,<notification-label> [<parameter-label>]*]*;
  ]
[REGISTERED AS object-identifier] ;
```

- BEHAVIOUR
  The BEHAVIOUR clause describes the behaviour of the Managed Object class as a whole and with respect to all its operations. It defines the conditions under which the properties of the Managed Object may be altered by either internal stimuli (operation of the Managed Object) or external stimuli (management action). Any CMIS protocol or notification which results from particular value is also specified. Behaviour is defined in natural language according to the rules of the OSI GDMO. Assertions (pre- and post-conditions for operations and invariants for inter-related Managed Objects) define behaviour and should be described in a more formal manner than natural language.

- ATTRIBUTE
  The ATTRIBUTE clause allows one or more attributes to be defined. The associated *propertylist* element allows optional specification of default, initial, permitted and required values, and of what operations on the attribute are allowed. Attributes are defined by means of the ATTRIBUTE template shown below:

```
<attribute-label> ATTRIBUTE
  DERIVED FROM <attribute-label> |
  WITH ATTRIBUTE SYNTAX type-reference
  [MATCHES FOR     qualifier [, qualifier]*;
  ]
  [BEHAVIOUR       <behaviour-definition-label>
                  [,<behaviour-definition-label>]*;
  ]
  [PARAMETERS     <parameter-label> [, <parameter_label>]*;
  ]
[REGISTERED AS object-identifier] ;
```

  The attribute template may be defined either independently, or derived from another template. In the latter case the DERIVED FROM clause allows the specification of another attribute template from which the currently defined template is derived.

The WITH ATTRIBUTE SYNTAX specifies the ASN.1 data type of the attribute.

The MATCHES FOR clause specifies the rules by which assertions about the value of an attribute are evaluated. Allowed rules are EQUALITY, ORDERING, SUBSTRING, SET-COMPARISON and SET-INTERSECTION. If no MATCHES FOR clause is specified, then no matching rules are supported for the attribute.

- ATTRIBUTE GROUPS
  The ATTRIBUTE GROUPS clause allows a collection of attributes to be referred to using a single name. The ATTRIBUTE GROUP template is shown below:

```
<group-label> ATTRIBUTE GROUP
    [GROUP ELEMENTS <attribute-label> [, <attribute-label>]*;
    ]
    [FIXED ;
    ]
    [DESCRIPTION  delimited-string ;
    ]
 REGISTERED AS object-identifer;
```

  The GROUP ELEMENTS clause specifies the attributes that constitute the attribute group being defined. If present, the FIXED clause indicates that the attribute group is defined to be of fixed membership. The DESCRIPTION clause describes the semantics of the group to be specified.

- ACTIONS
  The actions clause allows the definition of the behaviour and syntax associated with a particular action type. Action types defined by the ACTION template may be used in the M-ACTION Management Service primitive. The template also specifies if the action is confirmed or unconfirmed.

- NOTIFICATIONS
  This clause allows a list of notifications that may be reported by the Managed Object to be defined. Notifications are defined by means of a template that specifies the behaviour and syntax associated with a particular Notification type.

4.3.5.3 Name Binding

OSI define an additional template (the NAME-BINDING template) that allows a Managed Object to be named. It defines an attribute as a naming attribute for a Managed Object class and specifies the superior Managed Object class with which a MANAGED OBJECT CLASS instance can form a containment relationship.

It is really the Name Binding that defines the create operation and behaviour, not the Managed Object Class. The same Class can have different Name Bindings with different Create / Delete operations defined, and different behaviours for them. For example, Create may be permitted in one Name Binding but not in another.

4.4 Internet Management Models

The Internet Activity Board (IAB) has recommended that all Internet Resources should be remotely manageable using SNMP. This requires use of the Internet Structure of Management Information (SMI) to define Managed Objects, as described in RFC 1155 (SNMPv1) and RFC 1442 (SNMPv2).

The Internet SMI describes how to define management information contained in a Management Information Base (MIB) for use by SNMP. A MIB has been defined for some common Internet Resources in RFC 1213 (see reference **MIB-II**).

The Internet SMI has initially been applied to network Resources, but can naturally be expanded to cover distributed system Resources.

Earlier IAB attempts to apply the OSI Management model to manage Internet Resources (RFC 1189 — CMOT, and RFC 1214 — the OIM MIB-II) have since been deprecated. More recently, the Network Management Forum ISO / Internet Management Coexistence (IIMC) working group has defined an ISO / CCITT GDMO translation of the Internet MIB-II; examples are illustrated in Section A.2.

4.4.1 Internet SMI

4.4.1.1 Objects

The Internet SMI specifies the format of SNMP Managed Objects. The Internet SMI does not use object-oriented terminology or concepts to the same extent as GDMO and related standards. It talks in terms of *object types* rather than classes. These *object types* are similar to OSI *attribute types*. Sets of these object types are collected into a Management Information Base (MIB) for a specific managed Resource (for example, the FDDI MIB).

Each type of object (termed an object type) has a name, a syntax and an encoding. The name is represented uniquely as an *object identifier*. The syntax for an object type defines the abstract data structure corresponding to that object type. For example, the structure of a given object type might be an *integer* or *octet string*. The encoding of an object type is simply how instances of that object type are represented using the object's type syntax. Implicitly tied to the notion of an object's syntax and encoding is how the object is represented when being transmitted on the network.

To avoid confusion, an *SNMP Object* is referred to as a *variable*, which is a synonym for *SNMP Object* that is commonly used in the Internet community.

Syntax

The SNMPv1 Internet SMI (RFC 1155) uses the ASN.1 syntax to specify variables. However, only the following restricted subset of ASN.1 is permitted:

- Only the ASN.1 primitives INTEGER, OCTET STRING, OBJECT IDENTIFIER and NULL are permitted; BOOLEAN, REAL, ENUMERATED, BIT STRING and character strings are not permitted. A string of printable characters is expressed as an OCTET STRING (MIB-II uses the definition:

  ```
  DisplayString ::= OCTET STRING
  ```

  for printable strings).
- Enumerated INTEGERs are permitted, but the value 0 is reserved, so must not be present in the list of enumerations.

- Of the ASN.1 structured types, only SEQUENCE and SEQUENCE OF are permitted.
- Application-wide types may be defined provided they resolve into an IMPLICIT-ly defined ASN.1 primitive type, a SEQUENCE or some other application-wide type.

The SNMPv2 Internet SMI (RFC 1442) adds support for additional ASN.1 features such as full ASN.1 subtyping and 64-octet counters. A new SNMPv2 construct called ‘‘Textual Conventions’’ (RFC 1443) is defined to facilitate reuse and display of common complex ASN.1 types.

Operations

SNMP models all management functions as alterations or inspections of variables. SNMP defines the following operations that can be performed on variables:

Get Request	This message is used by a managing system to request the values of one or more specified variables from an Agent system.
Get Next Request	This message is used by a managing system to request the values of objects that are the immediate successors of one of more specified variables.
Set Request	This message is used by a managing system to set the values of one or more variables in an agent system.
Get Response	This message is used by an agent system to indicate failure or to return data to a managing system in response to a Get Request or Get Next Request message or to indicate success / failure of a Set Request operation.
Trap	This message is used by an agent system to deliver unsolicited messages to a managing system. The SNMP specification includes 6 types of specific traps, plus the *Enterprise Specific* type that can be used to indicate events that do not fall into the standard set of traps. The SNMPv1 traps are listed below: • Cold Start • Warm Start • Link Down • Link Up • Authentication Failure • EGP Neighbour Loss • Enterprise Specific

SNMPv2 (RFC 1448) defines two additional messages: Get-Bulk (an automatically repeating Get-Next) and Inform-Request (roughly analogous to a confirmed Trap which includes data, or an unsolicited Get-Response). SNMPv2 SMI (RFC 1442) defines an additional access value ‘‘read-create’’ which can be used to instantiate or erase conceptual table entries (roughly analogous to OSI create and delete operations). No equivalent functionality exists in ANMPv1. Note that there is no equivalent to the PSI action; this can be implemented by use of variables which when set cause an action to occur.

The SNMP protocol does not support the Scoping and Filtering functions of CMIP. However, a single SNMP message can specify a list of variables as its target, and the Get-Next or Get-Bulk Request can be used to traverse parts of the Internet MIB.

Notifications

The Internet SMI does not formally associate events with managed Resources. Whereas GDMO includes the definition of notifications (that is, events), in the Managed Object class definition, the Internet SMI format does not include notifications in variable type definitions. However, notifications are still supported and, as described above, the SNMP Trap and Inform-Request messages are used to transmit them.

Definition

The Internet variable type format includes a *definition* section. This is equivalent to the GDMO *behaviour* section and allows a natural language description of the semantics of the variable type.

4.4.1.2 Structure/Relationships

The Internet SMI does not have all the structuring mechanisms of GDMO. The only structuring mechanisms available are:

Variable Groups	The Internet SMI aggregates variable definitions into groups. A group is the basic unit of conformance, that is, if the semantics of a group are applicable to an implementation then all objects of the group must be implemented.
Tables	As discussed earlier, the Internet SMI allows the ASN.1 structured type SEQUENCE OF to be used in the definition of object types. This allows variables to be defined that contain instances of other variables (similar to the containment relationship in GDMO). These are referred to as tables, an example being a routing table which consists of a sequence of route entries.

4.4.1.3 Internet Managed Object Type Macro

Whereas GDMO expresses the format of Managed Object classes through the definition of templates, Internet SMI defines an OBJECT-TYPE Macro to aid in the definition of variable types. The much simpler structure of Internet variables discussed above is reflected in a fairly simple OBJECT-TYPE macro. The macro allows the following to be defined for SNMPv1:

Object Descriptor	A textual name (value reference) for the variable.
Object Identifier	As in GDMO, an administratively assigned unique name that can be used in the protocol to identify the variable.
Syntax	The abstract syntax for the variable type.
Definition	A textual description of the semantics of the variables type.
Access	One of read-only, read-write, write-only or inaccessible.
Status	One of mandatory, optional or obsolete.

A slightly extended OBJECT-TYPE macro is defined for SNMPv2 (see reference **SMIv2**). SNMPv2 also defines MODULE-IDENTITY and NOTIFICATION-TYPE macros for use when defining MIBs.

4.4.1.4 Extending the Internet MIB

The Internet SMI does not support inheritance, hence this mechanism cannot be used to extend the MIB. Instead, the following extensibility mechanisms must be used:

- the addition of new standard variables through the definition of new versions of an existing MIB (for example MIB-II RFC 1213 which supersedes MIB RFC 1158) or the creation of a new MIB. New versions may declare old variables obsolete (but not delete their names), augment the definition of list variables by defining additional members, and define entirely new objects. New versions may not change the semantics of previously-defined variables without changing their names.
- the addition of widely-available but non-standard variables through the experimental subtree. The Internet MIB has four nodes at the top of the naming hierarchy: directory, management, experimental and private. All Managed Objects defined in IAB recommended RFCs will occur under the management subtree (see Chapter 7).
- the addition of private objects through the enterprise subtree. The enterprise subtree is a node allocated immediately under the private node.
- in SNMPv2 SMI, Textual Conventions and the AUGMENTS clause have been added to facilitate reuse and extensibility.

4.5 Mapping Between Object Models

4.5.1 Mapping Between OMG Objects and ISO Managed Objects

Currently the mapping between OMG objects and ISO Managed Objects is not subject to any standard defined process. Several comparison studies have been performed, and this topic is expected to be addressed as part of the X/Open Systems Management Programme.

4.5.2 Mapping Internet Managed Objects to ISO Format

Currently, a mapping between Internet objects and ISO Managed Objects is being defined by the Network Management Forum ISO/Internet Management Coexistence (IIMC) working group. IIMC specifications include both a mapping of Internet MIBs into ISO GDMO template format, and a mapping of ISO MIBs into Internet SNMPv1 and SNMPv2 macro format. These mappings are intended to preserve the syntax and semantics of the source MIB in the target format. An example of the Internet MIB-II translated into ISO GDMO template format is provided in Section A.2.

Further information relating to the issues of the coexistence of ISO and Internet management is contained within another X/Open Guide (see reference **COIW**).

4.6 Refinement

One of the key benefits of the use of object-oriented techniques is the ability to "refine" an existing object definition to create a new, modified definition, thus re-using the work embodied in the original definition and implementation.

There are 3 primary techniques by which this is achieved, Inheritance, Allomorphism, and Polymorphism.

4.6.1 Inheritance

Inheritance is the means by which a new object definition *inherits* all the properties of the existing definition. Additional properties are then added to the definition to create a new definition which has all the original properties plus the new additions.

4.6.2 Polymorphism

Polymorphism (literally ‘‘many forms’’) is a mechanism by which an object class can modify the behaviour it inherits. It provides a mechanism by which an object can be used exactly as if it were an instance of another class. The behaviour of the object will depend on the context in which it is used.

4.6.3 Allomorphism

Allomorphism (literally ‘‘other form’’) provides essentially the same facility as polymorphism. The term was invented, however, to break away from the assumption that there would be some inherited implementation. As long as the implementation of an object interface conforms to its definition, (and encapsulation is complete), there should be no way to tell whether the object’s implementation is separate or shared with that of another class.

This view of allomorphism as interface compatibility is particularly relevant when considered in the context of distributed systems. When a ‘‘remote’’ object can only be accessed via some standard interface, the implementation techniques used to provide that interface on the remote side are irrelevant provided that the object ‘‘does the right thing’’ in response to any interaction.

Chapter 5

Searching for Managed Objects

A large number of organisations are contributing to Systems Management. They have all adopted an object-oriented approach to defining the functionality they wish to see in the area of Systems Management.

This chapter identifies some relevant organisations, and briefly describes their activities.

Appendix B lists some existing ISO and Internet MIBs.

ISO

Within ISO, the principal area of relevance is the definition of standards for both Layer Management and Systems Management, and the latter deals primarily with application level facilities for managing communications layers. However, the concepts and facilities are also applicable to Systems Management as defined in this document. ISO is primarily concerned with the definition of Managed Object classes relevant to communications systems.

ISO/IEC JTC1/SC21/WG4 is responsible for defining Systems Management standards and management support object classes, including generic OSI layer management information. A number of other working groups (including SC21, SC6) within ISO are responsible defining OSI layer Managed Objects and information models. For example, ISO/IEC 10165-2 defines management support objects such as Top, System, Event Forwarding Discriminator, and Log.

CCITT

CCITT (Comitée Consultatif Internationale de Télégraphique et Téléphonique) is an international consultative committee that produces international communications recommendations which are frequently adopted as standards or aligned with ISO standards. CCITT is a member of the International Telecommunications Union (a United Nations treaty organisation). CCITT recommendations in the area of Systems Management are in the X series (have an identifier of the form X.<number> - primarily in the X.700 series) and are technically aligned with ISO standards. (For example, CCITT Recommendation X.722 is technically identical to GDMO.)

A number of study groups within CCITT (including CCITT SG XV, XI and IV) are responsible for defining Managed Objects and information models. For example, CCITT M.3100 defines a Generic Network Model containing Managed Objects such as Network, Equipment, and Managed Element. Many national standards organisations such as ANSI T1 also define Managed Objects and information models which are often progressed internationally within CCITT study groups.

IEEE POSIX 1003.7

The POSIX System Administration working group (P1003.7) was formed in January 1989 with the goal of defining interfaces to systems administration functions in a network of heterogeneous systems. The group has subdivided into a number of subprojects each representing a functional subset of the area of interest of P1003.7, hence standards pertaining to specific areas of interest will be published. These will include Managed Object classes definitions.

Object Management Group (OMG)

The Object Management Group is a consortium that exists to define and promote a framework for distributed application integration based on object-oriented technology.

As part of their work they have announced an intention to establish an Interface Definition Registry.

Network Management Forum (NMF)

The NMF is an international consortium of information system and communication equipment suppliers, telecommunications service providers, and users. Focussed exclusively on management issues, the Forum seeks to accelerate the availability of solutions to industry-wide problems in network, systems, and service management. To that end, the Forum manages several programmes, including:

OMNI*Point*	A framework for service management that delivers comprehensive specifications for interoperability in multi-technology, multi-standard, multi-domain management environments. OMNI*Point* is a collaborative effort between the Forum, X/Open, and other consortia, government and commercial users, and standards bodies from around the world.
AIMS	A group of ISVs using fast prototyping to develop ways of integrating management applications.
SPIRIT	An effort by telecommunications service providers and their computing suppliers to define a procurement specification for a general purpose computing platform, compliant with XPG and OMNI*Point* that meets the needs of large-scale users.

OSE Regional Workshops

The primary aim of the OSE regional workshops is the development of ISO International Standardised Profiles (ISPs). There are three workshops:

- European Workshop for Open Systems (EWOS). EWOS has established an Expert Group on network management.
- OSE Implementors Workshop (OIW). The main OIW group dealing with network management is the Network Management Special Interest Group (OIW NMSIG).
- Asia and Oceania Workshop for Implementors of OSE (AOW).

All three are contributors to the proposed Draft ISPs in the area of OSI management.

In addition to the above, there are other workgroups that are defining Managed Object classes to reflect their special needs.

Open Software Foundation (OSF)

OSF is addressing the area of distributed Systems Management. It has defined a Distributed Management Environment (DME). The OSF Management Special Interest Group is defining DCE, DME, and Operating System Managed Object classes.

UNIX International

UI has established a working group to develop a model for distributed management and a detailed set of requirements for an open management framework. They have developed a distributed object model for Systems Management covering:

- Framework Common Utilities: these are drawn from the UI Atlas Distributed Computing Architecture, and are centred on a distributed object management facility. They support distributed facilities that are common to many application domains.
- Application Objects: these define objects that can be used in designing APIs. Each object encapsulates a particular aspect of management functionality.
- Application Presentation Layer: this defines a common object-oriented API for human interaction.
- Application Programming Interface: this defines a common object-oriented API to allow applications to access Management Services in a location-independent manner.

Internet Engineering Task Force (IETF)

The IETF is one of the principal subsidiary bodies of the Internet Activities Board (IAB). The IAB has overall responsibility for Internet engineering and management. The IETF responsibilities include the specification of short and mid-term Internet protocols and architecture, and recommending standards for IAB approval. It is organised into a number of technical areas which include network management and OSI coexistence. The IETF has adopted the object model in order to specify the Internet network management information base and protocols.

There are a large number of working groups within the IETF which are chartered to produce Internet MIBs. For example, the Host Resources MIB working group is chartered to define an Internet MIB which represents Resources of special interest to system administrators, such as operating systems, file systems, and running software processes. Appendix B identifies many other Internet MIBs developed by IETF working groups.

Chapter 6

Policy Freedom

Managed objects should be defined such that they do not unnecessarily impose a usage policy on system administrators. The object-oriented systems management model can impose policy through:

- the definition of Managed Object classes. The existence of a Managed Object corresponding to a Resource itself imposes policy. For example, if a Managed Object corresponding to a Resource is defined such that it includes a property *password*, this imposes the policy of requiring a password when the Resource is accessed. The definition of more general, uninstantiable objects may allow administrators to specialise specific objects that are more appropriate to their systems.
- the definition of initial, default, required and permitted values. Defaults and restrictions on the values of properties can further tighten the usage policy imposed by the existence of a Managed Object class definition.

Different instantiations of Managed Object classes will be subject to different usage policies because different administrators will have different policies. However, the complete absence of usage policies can be very intimidating to users. Hence vendors should be encouraged to ship ‘‘default policies’’, which can be altered by users to reflect local preferences.

By way of example, when an administrator wishes to add a user to a system, they need to specify the various properties of that user (such as login name, user-Id, group membership information, home directory, shell, etc.). However, defaults will be different on different systems; hence the Managed Objects representing the user should allow this freedom.

6.1 OSI Techniques

Within the OSI environment, the MIM and GDMO allow default and initial values to be specified in a number of ways:

- the Managed Object class or package definition can specify a fixed default value which may be assigned during object creation or in response to an M-Set(Replace-With-Default) operation,
  and
- an Initial Value Managed Object (IVMO) can be defined to store values which can be used to initialise attribute values during object creation.

In addition, it is possible to define the range of permitted values that an attribute may take.

6.2 Policy Objects

Another technique, used in at least one object-oriented systems management implementation, is to separate issues of policy into distinct policy objects. Such objects are used to enforce the locally defined policy.

Such a scheme is implemented by associating policy objects with the definition of an object class. There are 2 types of policy object:

- Policy Validation Objects which can be used to enforce a range of permitted values.
- Policy Defaults Objects which provide default initial values.

Chapter 7

Registration and Publication

7.1 OMG Object Registration

At the present time the OMG has announced an object registration scheme but it has not yet been implemented.

7.2 ISO/Internet Registration

The process of defining Managed Object classes requires the assignment of globally unique identifiers to Managed Object classes and to their components (for example, attributes). These identifiers are known as **object identifiers** and are carried in protocol, hence they are externally visible and must be globally unique. The IAB and ISO standards bodies have established a name registration procedure that can allocate unique object identifiers.

Managed object identifiers have a hierarchical structure, and an object identifier for a Managed Object class can be viewed as a sequence of labels that navigate through the object identifier tree to the Managed Object class. This tree consists of a root connected to a number of non-overlapping nodes, each identified by a *label*. A label is a pairing of a brief textual description and an integer (reference **SIMI**).

The hierarchical structure of object identifiers allows the global name space to be subdivided hierarchically, and responsibility for administrating specific sub-trees to be delegated to naming authorities (such as ISO, CCITT or IAB). A registration authority can assign individual object identifiers and/or delegate responsibility for sub-trees within their responsibility to other naming authorities. The hierarchical registration scheme actually covers all information objects (of which management information is only a small subset); examples are Managed Objects classes, FTAM document types, MHS security categories and Internet management objects.

A registration authority has several responsibilities, which include ensuring the registration of unambiguous names, and providing any rules that subdomain naming authorities must comply with to meet the registration requirements.

The root node of the ISO/CCITT object identifier tree is itself unlabelled, but it includes three labelled nodes:

ccitt(0)	which is administered by the International Telegraph and Telephone Consultative Committee (CCITT)
iso(1)	which is administered by the International Organisation for Standardisation (ISO)
joint-iso-ccitt(2)	which is jointly administered by ISO and CCITT.

Each node has children of its own which are labelled. For example, under the iso(1) node, a subtree has been designated for use by other national and international organisations, org(3). Under the org(3) node the U.S. Department of Defense (DoD) has been assigned to administer a node. The entire path to this node has the following form:

```
dod    OBJECT IDENTIFIER ::= { iso(1) org(3) dod (6) }
               — commonly referred to as —
                          1.3.6
```

The IAB administers a node on behalf of the Internet community under the DoD subtree. At the topmost level the IAB has divided its subtree into four nodes, pertaining to directory, management, experimental and private use.

Directory	The directory subtree is for use of the OSI Directory in the Internet.
Management	The Management subtree (mgnt) is used to identify objects that are defined in IAB approved documents for the purposes of Internet network management.
Experimental	The experimental subtree is used to identify objects used in Internet experiments.
Private	The private subtree is used to identify objects defined unilaterally. This subtree has one child, *enterprise*, which is intended to allow parties providing networking subsystems to register models of their products.

It is important to be aware that that ISO and the IAB use identifiers in different ways. Within OSI management, object identifiers are assigned to object classes, attributes, notifications, and actions., but are not used to name instances of Managed Objects. Within Internet management, object identifiers are assigned to OBJECT-TYPE macros, and are also used to name instances of MIB variables. An Internet MIB variable is named by concatenating its OBJECT-TYPE object identifier with a value that uniquely names the instance (either zero, or conceptual table indices).

7.3 Publication

The Managed Object registration process involves the publication of Managed Object class definitions. Such Managed Objects are defined in the global space and, therefore, will be generally understood by open systems management applications (and administrators). X/Open Systems Management object classes and object classes referenced in systems management profiles will be published as public Managed Objects that define allomorphic Managed Object classes. Each hardware and software vendor should define Managed Object classes or packages that are derived through specialisation from this allomorphic set. In turn, each vendor-specific package should be registered and the definitions published so that the specialised attributes and behaviour can be interoperably supported in addition to the allomorphic attributes and behaviour.

However, it is not desirable that all Managed Object classes definitions should be widely published. Managed object classes that are created by end-users, or that are short-lived, or esoteric in nature, should be maintained as private Managed Objects. Private Managed Object classes are different from Managed Object classes that are registered by vendors as public Managed Object classes. Private Managed Object classes may be named and formally or informally registered with a local naming authority.

Public naming authorities are normally organisations such as the U.S. Department of Defense (DoD) or the Canadian Standardisation Authority (CSA). Private naming-authorities may be as diverse as a system administrator within a corporate organisation or university, or a central registration-authority for a conglomerate that shares Resources across a global wide area network. Private Managed Object classes can be specialised from existing public Managed Object classes, or created independent of the existing naming tree. In either case, the private Managed Object class should be assigned an unambiguous name to avoid name space collisions.

Creators of private Managed Object classes should be aware that unless a Managed Object class definition is made public, the system Manager will have to do something to integrate Managed Objects instantiated from it into a heterogeneous management system. In addition, should it ever become necessary or desirable to combine subdomains, private Managed Object classes can present reconfiguration problems if they are not carefully organised.

7.3.1 Catalogues and Repositories

An effective way of publishing a Managed Object definition is to have it included in a publicly available repository of definitions. In addition, a Managed Object can be included in a management information catalogue — a listing and cross-reference which identifies where Managed Objects are published.

The Network Management Forum maintains both a repository and a catalogue of Managed Object definitions. The repository, known as the NM Forum Library, accepts objects defined in GDMO that meet certain defined technical and business criteria (see references **NMFTC** and **NMFBC**). The Catalogue, republished with each OMNI*Point* dot release, provides an index of all Managed Objects included in the NM Forum Library, as well as many others which are under development throughout the industry. The Catalogue provides keywords and references which can be used to locate and obtain actual Managed Object definitions that may be of interest to those defining objects.

The Object Management Group has indicated an intention to maintain a repository of OMG-based object definitions. The repository will contain objects defined in IDL.

7.4 Guidelines for Publishing Objects

X/Open Systems Management recommends the following guidelines for deciding whether or not to publish Managed Object classes definitions:

1. A Managed Object class should be identified as useful for management of some Resource.
2. Evidence of current use and utility should be required.
3. To the extent possible, the number of Managed Object classes should be limited.
4. Redundant variations of Managed Object classes should be avoided; before defining a new Managed Object, published Managed Object class definitions should first be searched.
5. Local implementation-specific Managed Object classes should be excluded.

Managed object classes that meet these criteria should be registered through a naming authority.

Chapter 8

Naming

The process of registration allows Managed Object classes to be uniquely identified. A related process is *name binding*, which allows instances of Managed Objects (that is, instantiations of a Managed Object class) to be uniquely identified. OSI management uses a hierarchical naming scheme, while SNMP management uses a flat identification scheme. Names are used to obtain addresses; the simplest view is of a name-to-address translation mechanism. X/Open systems management must exist in an environment where multiple naming schemes are used.

8.1 OMG Naming

OMG objects are accessed by means of an object reference, which is a unique, immutable, opaque value, chosen by the implementation at object creation time. Object references are never re-used to refer to another object.

The way in which object references are associated with the names of Managed Objects will be the subject of additional object services.

8.2 OSI Management

In OSI systems management, the containment relationship is used to uniquely name instances of objects. Names are designed to be unambiguous in a specified context; for management this context is determined by the containing Managed Object. A subordinate Managed Object is named by the combination of:

- the name of its superior Managed Object
- information that uniquely identifies this Managed Object within the scope of its superior Managed Object. This information is called a **Relative Distinguished Name (RDN)**. An RDN consists of *one* attribute that is part of the Managed Object mandatory package. Attributes used for RDNs must be testable for equality, and their semantics must permit their values to be set when the Managed Object is instantiated, and to remain fixed for the lifetime of the Managed Object.

The naming process can be applied recursively, so that a Managed Object can be given a globally unique name called a **Distinguished Name**. A Managed Object's distinguished name consists of its RDN concatenated to the sequence of RDNs of each of its superior Managed Objects in descending order, starting at **root**. The top level of the naming tree (the root) is a null object (that is, the Managed Object has no associated properties) and always exists. For interoperability and portability it is necessary to limit the maximum length of Distinguished Names, and possibly of RDNs.

The particular attribute to be used to form an RDN of a Managed Object is specified in a name binding. A Managed Object class can have more than one name binding; however, an instance of the Managed Object class is created with a single naming attribute. A name binding identifies:

- the Managed Object class being named
- a Managed Object class, instances of which may contain instances of the Managed Object being named
- one or more attributes that will be used to form the RDN.

8.3 SNMP Management

Internet naming uses a flat identification scheme for naming MIB variables within the context of a given network address. Each Internet variable has a unique object identifier that can be used together with the network address of the system implementing the variable to uniquely identify a variable on a global basis. Object identifiers are assigned by registration authorities, as described in Section 7.2 on page 47.

Chapter 9

Conformance Testing

To ensure interoperability and portability, it is necessary to have the means of testing that published Managed Objects specifications have been implemented correctly. Such tests can take two forms:

Conformance Testing	These tests ensure that an implementation correctly obeys a published specification. To aid conformance testing, all OSI management-based specifications contain an Implementation Conformance Statement (ICS) proforma, which specifies conformance information in a tabular format. These ICS are divided into: • Managed Object Conformance Statements (MOCS); these are published in OSI standards, together with Managed Object definitions. • Protocol Implementation Conformance Statements (PICS); these are published in OSI standards together with the protocol specification.
Interoperability Testing	These tests involve the interconnection of systems from various implementors to demonstrate that they can interwork. Interoperability testing is undertaken privately or publicly as part of industry gatherings.

Conformance testing is normally undertaken by recognised bodies. In the area of Network Management, the NMF has defined conformance tests, while EWOS is defining a conformance test model which will describe a framework for object conformance testing. The current work of NMF concentrates entirely on conformance tests for network management based on OSI protocols. At the time of writing, no conformance testing for Internet network management exists.

The OMNI*Point* Testing Partners work includes conformance testing of both lower-level and higher-level protocols and of Managed Objects defined by OSI/NMF. They have not only specified conformance requirements for systems management, but have also endorsed or developed conformance testing facilities.

To be considered conformant, a product shall go through and pass a sequence of conformance tests. These tests are designed to provide an initial level of confidence that implementations will interoperate with minimum effort on the part of those concerned.

Appendix A

Examples

The examples in this section are based on the Internet MIB. This MIB has been translated into ISO / CCITT GDMO format, and also into OMG IDL.

Selected parts of the definitions, (based on the system group), are presented here for comparison purposes. In all cases, the examples below are not complete and are provided for illustrative purposes only.

A.1 Internet MIB Definitions

This section contains the interface and system groups drawn from RFC 1213, Management Information Base for Network Management of TCP / IP-based internets: MIB-II

A.1.1 The System Group

```
-- the System group

-- Implementation of the System group is mandatory for all
-- systems.  If an agent is not configured to have a value
-- for any of these variables, a string of length 0 is
-- returned.

sysDescr OBJECT-TYPE
   SYNTAX  DisplayString (SIZE (0..255))
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "A textual description of the entity.  This value
         should include the full name and version
         identification of the system’s hardware type,
         software operating-system, and networking
         software.  It is mandatory that this only contain
         printable ASCII characters."
   ::= { system 1 }

sysObjectID OBJECT-TYPE
   SYNTAX  OBJECT IDENTIFIER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The vendor’s authoritative identification of the
         network management subsystem contained in the
         entity.  This value is allocated within the SMI
         enterprises subtree (1.3.6.1.4.1) and provides an
         easy and unambiguous means for determining ‘what
         kind of box’ is being managed.  For example, if
         vendor ‘Flintstones, Inc.’ was assigned the
         subtree 1.3.6.1.4.1.4242, it could assign the
```

```
        identifier 1.3.6.1.4.1.4242.1.1 to its ‘Fred
        Router’."
    ::= { system 2 }

sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The time (in hundredths of a second) since the
        network management portion of the system was last
        re-initialized."
    ::= { system 3 }

sysContact OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The textual identification of the contact person
        for this managed node, together with information
        on how to contact this person."
    ::= { system 4 }

sysName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "An administratively-assigned name for this
        managed node.  By convention, this is the node’s
        fully-qualified domain name."
    ::= { system 5 }

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The physical location of this node (e.g.,
        ‘telephone closet, 3rd floor’)."
    ::= { system 6 }

sysServices OBJECT-TYPE
    SYNTAX  INTEGER (0..127)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A value which indicates the set of services that
        this entity primarily offers.

        The value is a sum.  This sum initially takes the
```

```
            value zero, Then, for each layer, L, in the range
            1 through 7, that this node performs transactions
            for, 2 raised to (L - 1) is added to the sum.  For
            example, a node which performs primarily routing
            functions would have a value of 4 (2^(3-1)).  In
            contrast, a node which is a host offering
            application services would have a value of 72
            (2^(4-1) + 2^(7-1)).  Note that in the context of
            the Internet suite of protocols, values should be
            calculated accordingly:

              layer  functionality
                1  physical (e.g., repeaters)
                2  data link/subnetwork (e.g., bridges)
                3  internet (e.g., IP gateways)
                4  end-to-end  (e.g., IP hosts)
                7  applications (e.g., mail relays)

            For systems including OSI protocols, layers 5 and
            6 may also be counted."
        ::= { system 7 }
```

A.2 MIB-II GDMO Definitions

This section contains definitions translated from those contained in RFC1213. The translation is quoted from an Internet Draft entitled ISO/CCITT and Internet Management Coexistence (IIMC): Translation of Internet MIB-II (RFC1213) to ISO/CCITT GDMO MIB (IIMCMIB-II).

This translation represents work in progress and is intended to illustrative. It does not provide the final, definitive translation.

A.2.1 IIMCMIB-II Managed Object Classes

```
internetSystem MANAGED OBJECT CLASS
  DERIVED FROM "Rec. X.721 | ISO/IEC 10165-2:1992":top;
      CHARACTERIZED BY
          internetSystemPkg PACKAGE
            BEHAVIOUR
          internetSystemPkgBehaviour BEHAVIOUR
      DEFINED AS
      !BEGINPARSE
      REFERENCE !!This Managed Object class maps to the
      Internet system group with object id {mib-2 1} in
      RFC 1213. See RFC 1213 for attribute semantics.!!;

      DESCRIPTION !!When this object class is implemented in
      a managed system for use with the ISO/CCITT management
      protocol (CMIP), this object class shall emit the
      internetAlarm notification in place of SNMP
      traps/notifications which are reported using the
      unconfirmed service, and in place of InformRequests
      which are reported using the confirmed service.
```

```
        When this object class is implemented in an ISO/CCITT-
        Internet proxy, the internetAlarm shall be emitted upon
        receipt of SNMP traps/notifications which are reported
        using the unconfirmed service, and emitted upon receipt
        of InformRequests which are reported using the
        confirmed service.!!;
        ENDPARSE!;;
        ATTRIBUTES
        {iimcManagementDocMan 1}: internetClassId GET,
          sysDescr          GET,
          sysObjectId       GET,
          sysUpTime         GET,
          sysContact        GET-REPLACE,
          sysName           GET,
          sysLocation       GET-REPLACE,
          sysServices       GET;
        NOTIFICATIONS
        {iimcManagementDocMan 1}:internetAlarm;;;
REGISTERED AS   {iimcAutoTrans 1 3 6 1 2 1 1};
```

A.2.2 IIMCMIB-II Attributes

```
sysContact ATTRIBUTE
    DERIVED FROM {iimcManagementDocMan 1} :displayString;
    BEHAVIOUR
        sysContactBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysContact with object id
        {system 4} in RFC1213. See RFC 1213 for attribute
        semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 4};

sysDescr ATTRIBUTE
    DERIVED FROM {iimcManagementDocMan 1} :displayString;
    BEHAVIOUR
        sysDescrBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysDescr with object id
        {system 1} in RFC1213. See RFC 1213 for attribute
        semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 1};

sysLocation    ATTRIBUTE
    DERIVED FROM {iimcManagementDocMan 1} :displayString;
    BEHAVIOUR
        sysLocationBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysLocation with object
        id {system 6} in RFC 1213. See RFC 1213 for
        attribute semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 6};

sysName       ATTRIBUTE
    DERIVED FROM {iimcManagementDocMan 1} :displayString;
    BEHAVIOUR
        sysNameBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysName with object id
        {system 5} in RFC1213. See RFC 1213 for attribute
        semantics. Usually the node's domain name.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 5};

sysObjectId    ATTRIBUTE
    WITH ATTRIBUTE SYNTAX
        IIMCRFC1213ASN1.ObjectIdentifier;
    MATCHES FOR            EQUALITY;
    BEHAVIOUR
        sysObjectIdBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysObjectId with object
        id {system 2} in RFC1213. See RFC 1213 for
        attribute semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 2};

sysServices    ATTRIBUTE
    WITH ATTRIBUTE SYNTAX IIMCRFC1213ASN1.Integer;
    MATCHES FOR  EQUALITY, ORDERING;
    BEHAVIOUR
        sysServicesBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysServices with object
        id {system 7}. See RFC 1213 for semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 7};

sysUpTime      ATTRIBUTE
    DERIVED FROM {iimcManagementDocMan 1}: timeTicks;
    BEHAVIOUR
        sysUpTimeBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        REFERENCE
        !!This attribute maps to sysUpTime with object id
        {system 3} in RFC1213. See RFC 1213 for attribute
        semantics.!!;
        ENDPARSE!;;
REGISTERED AS {iimcAutoTrans 1 3 6 1 2 1 1 3};
```

A.2.3 IIMCMIB-II Name Bindings

```
internetSystem-systemNB  NAME BINDING
    SUBORDINATE OBJECT CLASS    internetSystem
                        AND SUBCLASSES;
    NAMED BY SUPERIOR OBJECT CLASS
        "Rec. X.721 | ISO/IEC 10165-2 : 1992" :system
                        AND SUBCLASSES;
    WITH ATTRIBUTE
            {iimcManagementDocMan 1}: internetClassId;
    BEHAVIOUR
        internetSystem-systemNBBehaviour BEHAVIOUR
        DEFINED AS
        !BEGINPARSE
        DESCRIPTION
        !!The <internet instanceId> portion of
        the internetClassId value shall be 0.!!;
        ENDPARSE!;;
REGISTERED AS {iimcManagementNB 1 3 6 1 2 1 1 };
```

A.2.4 IIMCMIB-II ASN.1 Module

```
IIMCRFC1213ASN1 {iimcManagementModAuto 1213 1354}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
IMPORTS  iimcManagementDocAuto, iimcManagementModAuto,
        iimcAutoTrans, iimcManagementNB,
        FROM IimcAssignedOIDs {iimcManagementModMan 1};

-- The following registration identifier is assigned to this
-- document using procedures defined in [IIMCIMIBTRANS]:

iimcMIBII OBJECT IDENTIFIER ::=
            {iimcManagementDocAuto 1213 1354}

-- Generic syntax

Integer ::= INTEGER

OctetString ::= OCTET STRING

ObjectIdentifier ::= OBJECT IDENTIFIER
```

```
-- MIB specific syntax

Integer128    ::= INTEGER (0..127)

Integer64k    ::= INTEGER (0..65535)

END
```

A.3 OMG IDL Definitions

This section contains OMG IDL definitions corresponding to the SNMP interfaces and systems groups. In addition, it includes a C language header containing supplementary definitions of data types.

This translation represents work in progress and is intended to be illustrative. It does not provide the final, definitive translation.

A.3.1 <mib2.h>

```
/*
 * Description:
 *  C header file for mib2 IDL objects
 *
 * SNMP MIB-II mandatory objects
 *  system
 *  interfaces
 *  ip
 *  icmp
 *  snmp
 *  objectControl - imposed by this interface
 *
 * SNMP MIB-II optional objects
 *  ucp
 *  tcp
 *  egp
 */

/*
 * General Use Types
 */
typedef unsigned long   Counter_t;
typedef unsigned long   Gauge_t;
typedef string<256>     DisplayString_t;
typedef sequence<octet,128> OctetString_t;
typedef unsigned long   TimeTicks_t;
typedef OctetString_t   ObjId_t;
typedef OctetString_t   IpAddress_t;

typedef unsigned long   Index_t;
typedef unsigned long   TableId_t;
typedef unsigned long   EntryId_t;
typedef void             STATUS
```

A.3.2 mib2-control.idl

```
/*
 * objectControl
 */
interface mib2-control {

            /*
             * Attributes
             */
            security
            commAttr
}
```

A.3.3 mib2-system.idl

```
/*
 * Description:
 *  IDL specification file for SNMP MIB II system group
 */
interface mib2-system {

   /*
   **
   ** Attributes
   **
   */
   readonly attribute DisplayString_t    sysDescr;
   readonly attribute ObjId_t            sysObjectId;
   readonly attribute unsigned long      sysUpTime;
   readonly attribute DisplayString_t    sysContact;
   readonly attribute DisplayString_t    sysName;
   attribute DisplayString_t             sysLocation;
   readonly attribute short              sysServices;    // 0..127
}
```

Currently Existing MIBs

Currently, ISO / CCITT and Internet-based object definitions are available or underway for a large number of Resource technologies, some of which are shown in the table below.

All of the MIBs listed are in the public domain, even when modeling proprietary Resources.

ISO/CCITT GDMO-based libraries		**Internet SMI-based MIBs**
10165-2 | X.721 - Definition of Management Information		MIB-II
M.3100 - Generic Network Model		Token Bus MIB
802.3H - Hub Management		Token Ring MIB
ANSI FDDI		DS1 MIB
Forum R1 Library		DS3 MIB
CNMA Library		Appletalk MIB
IDRP		OSPF MIB
ES-IS, IS-IS Routing		BGP v3 MIB
G.784 - SDH		Remote Network Monitoring MIB
T1.214		Ether-like MIB
T1.215		FDDI MIB
IETF OIM MIB-II		Bridge MIB
ETSI Traffic Model		DECnet Phase IV MIB
ETSI Transmission Equipment		SMDS Interface Protocol MIB
10165-5 - Generic Management Information		RS-232-like MIB
10733 - OSI Transport Layer		Printer MIB
10737 - OSI Network Layer		Character Stream MIB
OSI Data Link, Physical Layer		X.25 MIB
OSF DCE Objects		Frame Relay MIB
OSF DME Objects		IS-IS MIB
OSF / 1 Objects		Chassis MIB
CCITT SG XV SONET Objects		OSI IP MIB
P1003.7 System Administration Objects		Host MIB
Forum OMNI*Point* Libraries		PPP MIB
OIW OMNI*Point* Libraries		IDRP MIB
IEEE 802.x Objects		Ethernet Repeater MIB
Q.751 SS7 Objects		IP Forwarding MIB
Q.94x ISDN Objects		Token Ring Repeater MIB
T1X1.5 SONET Objects		

Glossary

AOW
Asia and Oceania Workshop for implementors of OSE.

ASN.1
Abstract Syntax Notation One.

CD
ISO Committee Draft.

DIS
ISO Draft International Standard.

EWOS
European Workshop for Open Systems

GDMO
Guidelines for the Definition of Managed Objects.

Internet Activity Board
(IAB) The coordinating committee for Internet design, engineering and management. The body that sets Internet standards (RFCs) and thus standards for the IPS.

IDL
Interface Definition Language.

IEC
International Electrotechnical Commission.

IEEE
(The U.S.A. Institute of Electrical and Electronics Engineers) Organisation of engineers and engineering organisations that defines standards such as the 802 networking standard.

IETF
Internet Engineering Task Force. NMF ISO/Internet Management Coexistence working group.

IS
ISO International Standard.

ISO
International Standards Organization.

MIB
Management Information Base.

MIM
Management Information Model.

NMF
Network Management Forum.

NMSIG
Network Management Special Interest Group.

OIW
OSE Implementers Workshop.

OMG
Object Management Group.

OSE
Open Systems Environment

OSF
Open Software Foundation

OSI
Open Systems Interconnection.

RDN
Relative Distinguished Name.

SMI
Structure of Management Information.

SNMP
Simple Network Management Protocol - a protocol for managing IPS networks.

Index