OSI Systems Management, Internet SNMP and ODP/OMG CORBA as Technologies for Telecommunications Network Management

George Pavlou
Dept. of Computer Science, University College London
Gower Street, London WC1E 6BT, UK
<gpavlou@cs.ucl.ac.uk>

2.1. Introduction and Overview

In this chapter, we compare the ISO/ITU-T OSI Systems Management (OSI-SM) [X701], Internet Simple Network Management Protocol (SNMP) [RFC1157][RFC1905/6] and ODP/OMG Common Object Request Broker Architecture (CORBA) [CORBA] approaches to network, service and distributed applications management. The chapter also provides a tutorial overview of those technologies, but it assumes a basic understanding of data network and distributed system principles.

OSI and Internet management have adopted the manager-agent paradigm. Manageable resources are modeled by managed objects at different levels of abstraction. Managed objects encapsulate the underlying resource and offer an abstract access interface at the object boundary. The management aspects of entities such as Network Elements (NEs) and distributed applications are modeled through "clusters" of managed objects, seen collectively across a management *interface*. The latter is defined through the formal specification of the relevant managed object types or classes and the associated access mechanism, that is the management access service and supporting protocol stack. Management interfaces can be thought of as "exported" by applications in agent roles and "imported" by applications in manager roles. Manager applications access managed objects across interfaces in order to implement management policies. Distribution and discovery aspects are orthogonal to management interactions and are supported by other means. Both OSI and Internet management are primarily communications frameworks. Standardization affects the way in which management information is modeled and carried across systems, leaving deliberately unspecified aspects of their internal structure. The manager-agent model is shown in Figure 2.1. Note that manager and agent applications contain other internal objects that support the implementation of relevant functionality. Since these are not visible externally, they are depicted with dotted lines.

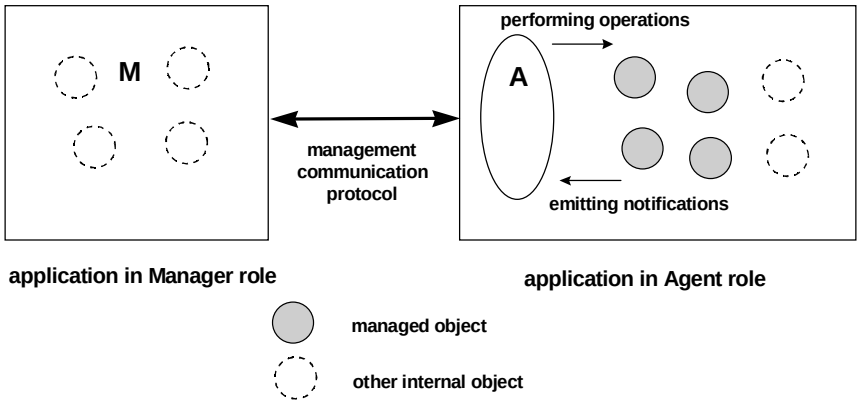


Figure 2.1 The Manager-Agent Model

The manager and agent roles are not fixed and management applications may act in both roles. This is the case in hierarchical management architectures such as the Telecommunications Management Network (TMN) [M3010]. In hierarchical management models, managed objects also exist in management applications, offering views of the managed network, services and applications at higher levels of abstraction. Management functionality may be organized in different layers of management responsibility: element, network, service and

business management according to the TMN model. Management applications may act in dual manager-agent roles, in either peer-to-peer or hierarchical relationships. Figure 2.2 shows three types of management organization according to the manager-agent model: centralized, flat and hierarchical. The centralized model is best exemplified by SNMPv1 Management Operation Centers (MOCs). The flat model reflects the evolution of SNMPv1 to SNMPv2, with "manager-to-manager" capabilities. Finally, the hierarchical model is best exemplified by the TMN, which uses OSI management as its base technology. Note that in both flat and hierarchical models, management applications are hybrid, assuming both manager and agent roles.

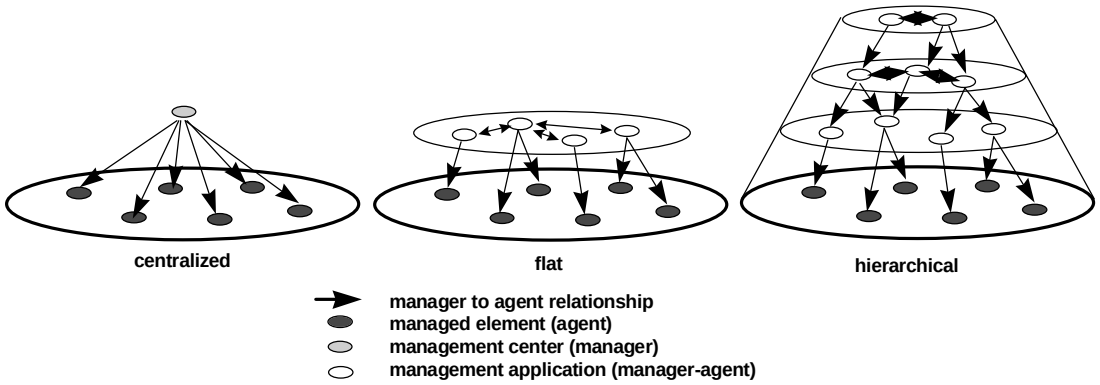


Figure 2.2 Models of Management Organization

Alhough OSI and Internet management conform to the same broad model, within their constraints there is room for wide divergence and, to an extent, they are at two ends of the spectrum. Internet management was designed to address the management needs of private data networks (LANs/MANs) and Internet backbones. As such, it has adopted a connectionless (CL), polling based, simple "remote-debugging" approach, opting for agent simplicity and projecting a centralized or flat organizational structure. On the other hand, OSI management was designed to address the management needs of OSI data networks and telecommunications environments. As such, it has adopted a connection-oriented (CO), event-driven, fully object-oriented (O-O) approach, opting for generality and trying to move management intelligence close to managed elements. It has been adopted as the base technology for TMN and it supports hierarchical organizational structures.

ISO/ITU-T Open Distributed Processing (ODP) [X901] is a general framework for specifying and building distributed systems. The Object Management Group (OMG) Common Object Request Broker Architecture [CORBA] can be seen as its pragmatic counterpart. While SNMP and OSI management are *communications* frameworks, ODP/OMG CORBA target a *programmatic* interface between objects in client or server roles and the underlying support environment, that is the Object Request Broker (ORB). Server objects are accessed through interfaces on which operations are invoked by client objects in a location transparent fashion. Choices made by the Internet Engineering Task Force (IETF) and ISO/ITU-T on the one side and OMG on the other side reflect their different pre-occupations: management communications for the former and distributed software systems for the latter. The difference in approach is sometimes referred to as "vertical" versus "horizontal" interfaces. Vertical interfaces standardize communications interactions between systems. The horizontal approach standardizes Application Programming Interfaces (APIs) which are used to "plug" application objects on the global supporting infrastructure. The latter is also referred to as the Distributed Processing Environment (DPE) and encapsulates the underlying network, hiding heterogeneity and providing various transparencies. The ODP / OMG CORBA model is shown in Figure 2.3.

The OMG CORBA paradigm is that of a client-server, with distribution provided through the ORB. The unit of distribution is the single object as opposed to the OSI and Internet object cluster that is visible across an interface. Client and server CORBA objects communicate through the ORB, whose services are accessed through standard APIs. Interoperability is achieved through the formal specification of server interfaces, the ORB APIs and the underlying inter-ORB protocols. One key difference to OSI and Internet management is that the object model and APIs have been addressed first, while the underlying protocols may be replaced. Of

course, interoperability dictates an agreed protocol but the rest of the framework is not heavily dependent on it. The key benefit is portability of objects across different CORBA implementations due to the standard ORB APIs and the various transparencies that are (or will be) supported by the latter. Note that communication aspects are "hidden" inside the ORB and, as such, are not shown in Figure 2.3. While OMG CORBA is a general distributed systems framework, its object-oriented nature and the fact that management systems are composed of interacting objects suggest that it could also be used for management. OMG CORBA has been chosen by the Telecommunication Information Network Architecture (TINA) [TINA] initiative as the basis for their DPE. TINA aims at a framework for future advanced services, including service and network management aspects.

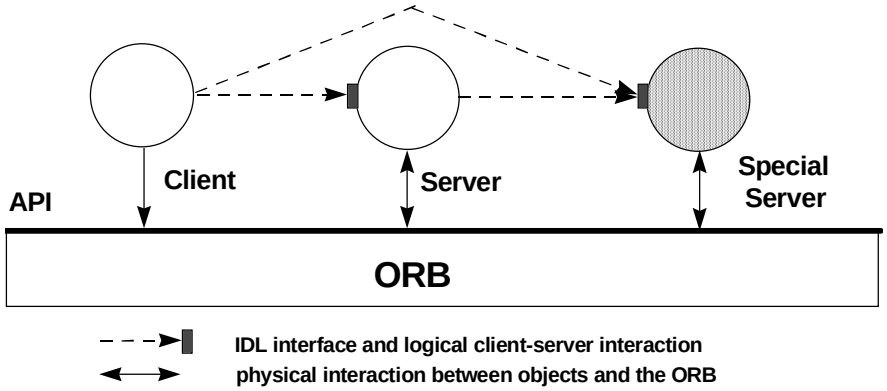


Figure 2.3 The OMG CORBA Model

In this chapter, we examine aspects of those technologies, assessing their suitability in the context of telecommunications management. In section 2.2 we examine information modeling aspects in the three frameworks and assess their flexibility and expressiveness. In section 2.3 we examine the access and distribution aspects, including the underlying communications and the paradigm of operation. In section 2.4 we examine other aspects, including applicability in various contexts, generic functionality and security. In section 2.5 we examine issues addressing their interworking and coexistence. Finally we present a summary and discuss potential future directions in telecommunication management.

2.1.1 Background Information: ASN.1

Some necessary background information concerns the OSI Abstract Syntax Notation 1 (ASN.1) [X208] language. This is an abstract "network" data structuring language that supports simple and constructed types. A particularly important ASN.1 type is the Object Identifier (OID), which expresses a sequence of non-negative integers on a global registration tree. OIDs are registered by standards bodies, for example ISO, ITU-T and IETF, and are used instead of friendly string names to avoid ambiguities. For example, the OSI *objectClass* attribute name is registered as {joint-iso-ccitt(2) ms(9) smi(3) part2(2) attribute(7) objectClass(65)}. ASN.1 is used in both the OSI management and SNMP frameworks to specify the management protocol packets and the structure of managed object information, for example attributes, operation parameters / results and notification information.

2.2. Management Information Models

A management framework and associated technology should be applicable to network, service, system and distributed application management. In addition, the applications and support infrastructure that constitute the management system should also be manageable. The ideal information model must cope easily with management information related to such a variety of management targets. At the same time, it must impose a measure of uniformity on the structure of management information so that it is possible to devise a set of generic management operations that are applicable in all management contexts. The original motivation for developing the three frameworks was different: network management for SNMP; network and service

management for OSI; and distributed application operation and management for OMG CORBA. As a result, the relevant information models, despite exhibiting similarities, have important differences.

A key difference between the SNMP and the OSI Management / OMG CORBA information models is that the latter have enthusiastically endorsed the O-O approach and have made full use of relevant concepts such as classes and inheritance. In the case of SNMP, the information model is often referred to as Object-based, with classes and inheritance regarded as unnecessary complications and thus deemed undesirable. All three models are quite general and, despite their differences, anything that can be modeled in one can also be modelled in another. In fact, methodologies have been developed for converting between them as described in section 5. The key issue is whether the greater expressive power one offers results to the better abstraction of management information and is worth the price of the additional complexity. Figure 2.4 shows a pictorial view of the notion of objects in the three frameworks and their collective view as a Management Information Base (MIB) across a management interface. Information modeling aspects are explained next, in a separate subsection for each framework.

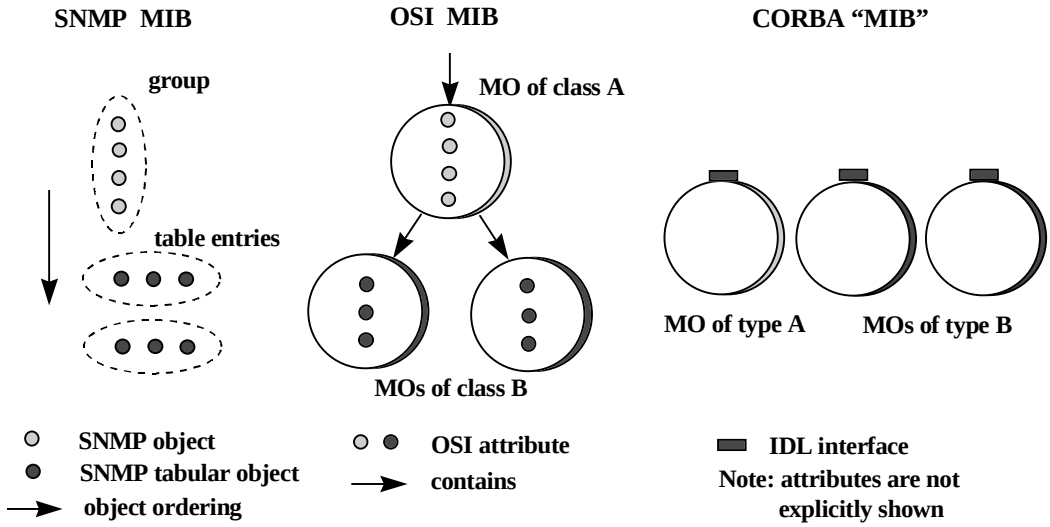


Figure 2.4 SNMP, OSI and CORBA Management Information Base

2.2.1 The SNMP Information Model

SNMP information modeling principles are collectively referred to as the Structure of Management Information (SMI) and are specified in [RFC1155] for SNMPv1 and in [RFC1902] for SNMPv2, the latter being an extension of the SNMPv1 model.

The basic building block of an SNMP MIB is the *object*. Objects belong to a particular *object type* and have *values*. According to the SNMP SMI, object values are restricted to a very small set of allowed syntaxes, resulting ultimately in the basic ASN.1 types INTEGER, OCTET STRING and OBJECT IDENTIFIER. Other *application-wide* types such as Counter, Gauge, NetworkAddress and Timeticks must resolve to either integer or string scalar types. The only constructed type allowed is a simple two-dimensional table, consisting of elements of the previous primitive types. Great emphasis is given to the fact that the allowable syntaxes are few, simple and scalar. The key advantage claimed for this approach is that object values carried across a network must be encoded in a "network-standard" way. Encodings and decodings can be computationally expensive, especially if the syntaxes involved are complex. Because SNMP uses only a small fixed set of syntaxes, it is possible to hand-code the encoding and decoding software in an optimal fashion. Thus, the SNMP SMI is analogous to a computer language that has a small set of basic types together with two-dimensional arrays.

The SNMP SMI defines a notation for specifying the properties of new object types, an ASN.1 *macro* called OBJECT-TYPE, while ASN.1 is used to specify the object syntaxes and the tabular structure. SNMP object-types can be either single or multiple instanced, with multiple instanced objects allowed only in tables. SNMP objects are similar to OSI Management / OMG CORBA attributes while there is no notion of a "composite" object-boundary that encapsulates a number of scalar objects modeling a manageable entity. The only composite relationship relates objects to tables.

A table has rows (also referred to as table entries or records), with each row represented by a *SEQUENCE* ASN.1 type that contains a statically defined number of objects. The table itself is modeled as *SEQUENCE OF* ASN.1 type with respect to the rows or entries, allowing an arbitrary number of those to be dynamically instantiated. A table thus resembles a "dynamic array of records". Tables may grow to arbitrary length but must always be of fixed width. A further rule restricts the syntaxes used within a row to be "scalar"; thus one cannot define tables within tables. Note also that tables and table entries are only conceptual composite objects in SNMP: only the individual scalar objects that constitute a table entry are accessible through the management protocol.

Let us examine the use of this modeling framework through an example. A typical single-instanced set of objects are, for example, those modeling aspects of a connection-oriented transport protocol entity such as the ISO Transport Protocol (TP) and Internet Transmission Control Protocol (TCP). Such objects will cover the number of current and previous connections, the number of incoming and outgoing unsuccessful connection requests, the number of transport packets sent, received and retransmitted and the number of various protocol-related errors. All these will have to be separate objects, loosely related through a "transport protocol" group. Note that this model does not support multiple instances of a transport protocol per node. If the latter was necessary, a table of transport protocol entries would be needed, but we will overlook this restriction for simplicity. The group will also comprise transport connections, which are multiple-instanced objects and have to be modeled through a table. A tpConnTable may be defined as a SEQUENCE OF tpConnEntry, with tpConnEntry being a SEQUENCE of objects modeling various aspects of the connection, such as source and destination access points, the connection state, and so on. Figure 2.4 shows objects of a single-instanced group, for example tp group, and two table entries, for example tpConnEntry. Note that both the group and table entries are depicted with dotted lines since there is no notion of a composite object boundary in the SNMP information model.

When objects are instantiated, they must be named so that they can be addressed unambiguously. The SNMP framework uses object identifiers for naming. Every *object type* has a registration OID, while an *object instance* is identified by the object type OID, suffixed by a part that identifies uniquely that instance. For non-tabular objects any suffix would do, so the minimal *.0* is used. For example, *tpCurrentConnections.0* is the instance of the object type *tpCurrentConnections*. In the case of multiple-instanced tabular objects such as the *tpConnState of the tpConnEntry*, the suffix needs to signify the table entry. The latter can be constructed from the values of one or more objects of that entry that constitute the "key", as specified in the relevant OBJECT-TYPE template for that table entry. In our example, the values of the *tpConnSrcAddr* and *tpConnDestAddr* objects may be used as they uniquely identify each connection. For example, *tpConnState.123.456* is the instance of the object type *tpConnState*, corresponding to the connection with source address 123 and destination address 456. Note that when object values are strings as opposed to integers, they need to be converted to object identifier suffixes: the SNMP SMI specifies rules for this conversion.

The SNMP naming architecture exhibits a very tight coupling between object type and instance identifiers. The problem with it is that instances of a particular object type can only appear in one particular place in the registration tree. This means that one cannot define generic object types to be used in several contexts. For example, a common object for representing the state as perceived by a managed resource is the *operationalState*. In the SNMP framework, it is not possible to define such a generic object but specific objects have to be defined for every particular context, for example *tpOperationalState*. An additional problem is that because this is not the intended use of OIDs, it has been found that most SNMP implementations spend a lot of processing time scanning through object names, trying to separate type and instance information. Finally, object names formed through OID suffixes are not natural.

SNMP objects accept only *Get* and *Set* operations. The Set operation can be performed only to objects that have a *read-write* or *read-create* access level according to the object specification. Access to objects with Get and Set operations is subject to the access control policy across a management interface. SNMP objects support neither *Create* nor *Delete* operations explicitly. Create and delete semantics are implicitly supported however for multiple-instanced objects, that is table entries or rows. In SNMPv1, row creation may be requested through a Set operation for an entry that is currently not in the table by setting the values of all the row objects. This type of Set operation is interpreted as table entry creation; however, this behavior is not mandated and implementations may behave differently. In addition, the SNMP protocol limits the maximum Protocol Data Unit (PDU) or packet size, so it might not be possible to pass values for all the row objects! SNMPv2 remedies this defect and uses an elaborate interaction scheme to ensure atomicity of row creation. This is supported through the use of a *RowStatus* object which must be present in any table that allows row creation. Creation can follow either a *createAndGo* or *createAndWait* protocol, according to the value of rowStatus. The former is similar to the SNMPv1 creation style and restricted by the maximum PDU size. In the latter, the values of the rowStatus object progress successively through *notReady -> notInService -> active* while the row is being created through more than one Set requests. Row deletion is achieved through a Set operation that sets the value of row status to *destroy*. Although all this works, it is certainly not simple. In fact, a high price has been paid in order to avoid introducing Create and Delete object operations through separate protocol primitives.

In both the OSI Management and OMG CORBA models, objects may accept arbitrary actions that operate on the object boundary, as described next. Given the fact that SNMP objects are essentially attributes compared to OSI and CORBA objects, imperative actions with arguments and results are meaningless. Nevertheless, actions are necessary and may be modeled in SNMP by objects that support the arguments and results of an action. An "action" may be emulated by a Set request, followed possibly by a Get request to retrieve the results. For example, a "reboot" action may be modeled by a boolean *rebootState* object whose value is set to *true* (in this case, there is no action result). This type of emulation is not very elegant and may result in complex interactions and an awkward object model for imperative commands with complex "argument and result" parameters.

In SNMPv1, agent applications may emit notifications called *traps* associated with the SNMP protocol rather than a MIB specification. These are supposed to be used only for a small and pre-defined set of events (*warmStart*, *coldStart*, *linkUp*, *linkDown*, *authenticationFailure*). However, some MIB designers (notably those of the Remote Monitoring MIB [RFC1271]) have extended the concept and have provided notations that allow MIB designers to specify resource-specific notifications and the information which should be included in them. A notation of this type has now been included in the SNMPv2 SMI [RFC1902].

2.2.2 The OSI System Management Information Model

Chapter 1 discussed OSI-SM and TMN information modeling in detail. The OSI-SM Information Model (MIM) is defined in [X720]. An OSI Management Information Base (MIB) defines a set of *Managed Object Classes* (MOCs) and a schema that defines the possible containment relationships between instances of those classes. There may be many types of relationships between classes and their instances, but containment is treated as a primary relationship and is used to yield unique names. The smallest re-usable entity of management specification is not the object class, as is the case in other O-O frameworks, but the *package*. Object classes are characterized by one or more mandatory packages while they may also comprise conditional ones. An instance of a class must always contain the mandatory packages while it may or may not contain conditional ones. The latter depends on conditions defined in the class specification. Managing functions may request that particular conditional packages are present when they create a managed object instance.

A package is a collection of attributes, actions, notifications and associated behavior. Attributes are analogous to object types in SNMP. Like an object type, an attribute has an associated syntax. Unlike SNMP, however, there are no restrictions on the syntax used. If desired, an attribute need not be of scalar type. A number of useful generic attribute types have been defined in [X721], namely *counter*, *gauge*, *threshold* and *tide-mark*. A MIB designer may derive resource-specific types from these. Support for arbitrary syntaxes provides a much

more flexible scheme than that of SNMP. For example, it allows the definition of complex attributes such as a *threshold*, whose syntax can include fields to indicate whether or not the threshold is currently active and its current value. It is also more expensive to implement since support for encoding and decoding of completely arbitrary syntaxes must be provided.

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

OSI Management follows a fully O-O paradigm and makes use of concepts such as inheritance. Managed object classes may be specialized through subclasses that inherit and extend the characteristics of superclasses. This allows re-usability and extensibility of both specification and associated implementation if an object-oriented design and development methodology is used. Pursuing the previous example, a *tpProtocolEntity* object class may inherit from an abstract *protocolEntity* class that models generic properties of protocol entities, such as the operational state and the service access point through which services can be accessed. By abstract class is meant a class that is never instantiated as such but serves only inheritance purposes. In the same fashion, an abstract *connection* class may model generic properties of connection-type entities such as the local and remote service access points, the connection state and creation and deletion notifications. The inheritance hierarchy of those classes is shown in Figure 2.5. It should be noted that conditional packages allow for dynamic (i.e. run-time) specialization of an object instance while inheritance allows only for static (i.e. compile-time) specialization through new classes.

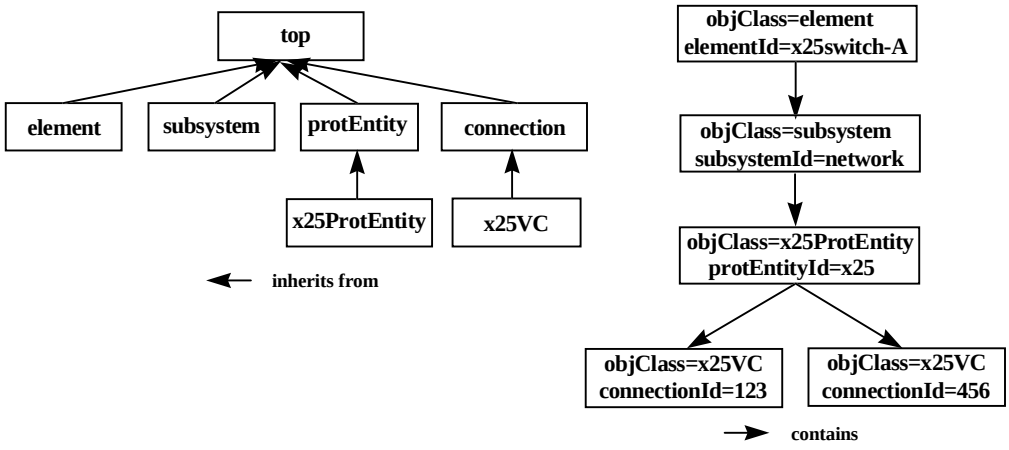


Figure 2.5 Example OSI Inheritance and Containment Hierarchies

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

In OSI management, a derived class may extend a parent class through the addition of new attributes, actions and notifications; through the extension or restriction of the value ranges; and through the addition of

arguments to actions and notifications. Multiple inheritance is also allowed and it has been used extensively by information model designers in standards bodies. Despite the elegant modeling that is possible through multiple inheritance, such models cannot be easily mapped onto O-O programming environments that do not offer such support, for example Smalltalk and Java. Multiple inheritance is a powerful O-O specification technique but increases system complexity.

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

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

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

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

The types of attributes, action and notification arguments, replies and exception parameters are specified as ASN.1 types. Object Identifiers are associated with classes, packages, attributes, notifications and actions but they have nothing to do with instance naming. Instead, managed object instances are named through a mechanism borrowed from the OSI Directory [X500]. Managed object classes have many relationships but containment is treated as a primary relationship to yield unique names. Instances of managed object classes can be thought as logically containing other instances. As such, the full set of managed object instances available across a management interface are organized in a Management Information Tree (MIT), also referred to as the "containment hierarchy". This requires that an attribute of each instance serves as the "naming attribute". The tuple of the attribute and its value form a Relative Distinguished Name (RDN), for example connectionId=123. This should be unique for all the object instances at the first level below a containing instance. If these instances belong to the same class, then it is the value of the naming attribute that distinguishes them (the "key").

The containment schema is defined by *name-binding* GDMO templates which specify the allowable classes in a superior/subordinate relationship and identify the naming attribute. Name bindings and naming attributes are typically defined for classes in the first level of the inheritance hierarchy, immediately under *top* so that they are "inherited" by specific derived classes. An example of a containment tree is shown in Figure 2.5, modeling connections contained by protocol entities, by layer subsystems, and by a network element. A managed object name, also known as a Local Distinguished Name (LDN), consists of the sequence of all the relative names from the top of the tree down to the object, for example *{subsystemId=network, protocolEntityId=x25, connectionId=123}*. OSI management names are assigned to objects at creation time and last for the lifetime of the object. An OSI managed object has exactly one name; that is the naming architecture does not allow for multiple names. The same is true for SNMP objects as described, though the naming architecture is different.

While SNMP was likened to a computer language with a few simple types plus arrays, OSI Management can be likened to a full object oriented language since it allows new types (of arbitrary complexity) to be defined and arbitrary methods (actions) to be invoked upon them.

2.2.3 The ODP / OMG CORBA Information Model

While both SNMP and OSI management are *communications* frameworks, standardizing management interfaces for applications in agent roles, OMG CORBA targets a *programmatic* interface between objects in client or server roles and the underlying support environment, that is the ORB. Server objects are accessed through interfaces on which operations are invoked by client objects.

The ODP / OMG CORBA information model is fully object-oriented, in a similar fashion to that of OSI Management. Objects are characterized by the *interfaces* they support. An ODP object may support multiple interfaces bound to a common state, unlike OSI management where objects may have only one interface. The current OMG specification, however, allows only a single interface per object. In fact, the OMG model defines objects through the specification of the relevant interfaces. As such, there is no direct concept of an object class in OMG. Object interfaces may be specialized through inheritance while multiple inheritance is also allowed. The root interface in the inheritance hierarchy is of type *Object*. OMG interfaces are specified using the Interface Definition Language (IDL) [IDL]. The IDL specification technique is more monolithic than the GDMO piecemeal approach: the minimum re-usable specification entity is the interface definition as opposed to the individual package, attribute, action and notification in GDMO. IDL may be regarded as broadly equivalent to the GDMO/ASN.1 combination in OSI management, though less powerful and with some differences highlighted below.

An OMG object may have attributes, accept operations at the object boundary and exhibit behavior. Such an object is used to implement a computational construct. In a management context, an object may behave as a manageable entity, modeling an underlying resource. Object attributes have associated syntax, which in IDL is called a *type*. Arbitrary syntaxes are allowed, although the expressive power of IDL types is less than ASN.1. There is no mechanism for grouping attributes together as in OSI management. Attributes accept Get and Set operations while only standard exceptions may signify an error during such operations. This is in contrast to

GDMO, where arbitrary class-specific errors and associated information may be defined to model exceptions triggered by attribute-oriented operations. OMG objects also accept object-oriented operations, similar to the GDMO actions. The normal execution of an operation results in a reply while object-specific exceptions may be defined. Operation invocations, replies and exceptions may take arbitrary parameters in terms of IDL types. It should be mentioned that a GDMO action may result in multiple replies, despite the fact information model designers seldom use this feature. Multiple results are not explicitly supported in IDL but may be modeled through "callback" invocations.

A key difference between GDMO and OMG objects is that the latter do not allow for the late binding of functionality to interfaces through optional constructs similar to the GDMO conditional packages. An OMG object type is an absolute indication of the characteristics of an instance of that type. However, attribute and operation parameter values may be "null" while CORBA supports a standard *not_implemented* exception. An additional major difference is that in IDL it is not possible to specify event types *generated* by an object: events are modeled as "operations in the opposite direction". As such, events are specified through operations on the interface of the receiving object. An OMG managed object needs to specify a separate interface containing all the events it can generate; the latter needs to be supported by managing objects that want to receive these events. There are more differences with respect to the way events are disseminated, but these are discussed in section 2.3.

ODP / OMG do not provide a built-in operation for instantiation of interfaces by client or managing objects. The reason for that is that OMG takes a "programmatic" view of object interfaces and, as such, a create operation is meaningless before that interface exists! While GDMO objects appear to accept create operations according to the specification, the latter are essentially targeted to the agent infrastructure in engineering terms. As such, interface creation in OMG may only be supported by existing interfaces: *factory* objects may be defined that allow client objects to create application specific interfaces. This approach is not flexible as a factory interface is necessary for every other interface that can be dynamically created. A more generic *factory service* would be welcome, allowing flexibility in the placement of new objects as currently factory objects may place new objects in the same node.

Deletion of objects is possible through the OMG Object Life-Cycle Services [COSS]. The latter has specified an interface that provides a *delete* as well as *copy*/*move* operations. Any other interface that needs to be deleted should inherit from the life-cycle interface. The copy/move operations apply to object implementations and appear to be very powerful as they support relocation and replication. The downside is that it is not at all clear how these will be provided. In the absence of implementations supporting life-cycle services at present, interface deletion is currently tackled through the definition of interface-specific *delete* operations. The problem is that if an object receives a delete request through its interface and deletes itself, there can be no reply to the performing client. An exception instead is raised and the client will never know if deletion was completed successfully or something else went wrong while the object was cleaning-up its state. Hopefully mature implementations of the life-cycle service interface will solve such problems in the future.

In summary, creation and deletion of interfaces is not handled in a fully satisfactory fashion. The main problem is that such facilities should not be seen as separate "services" but should be an integral part of the underlying platform, that is the ORB. Unfortunately, the OMG did not take this approach. Finally, it should be mentioned that object creation and deletion in distributed system contexts are used mostly for system instantiation and termination, that is not very frequently while this is *not* the case in management environments.

While the OMG IDL object model has many similarities to GDMO, a marked difference concerns naming. OMG objects can be identified and accessed through *Object References*. The latter are assigned to objects at creation time and are opaque types, that is have no internal structure and, as such, do not reveal any information about the object. Their typical implementation is through long bit-strings in order to facilitate processing; they are in fact similar to pointers in programming languages. An object may have more than one reference while objects may also be assigned names. The latter are distinct from objects, unlike Internet and

OSI management where an object always has a name. Actually OMG objects need not have names at all as they may be "bound to" by type through the ORB and accessed through their interface reference(s). In addition, names may be assigned to objects but this mapping may change at any time. Names are assigned to objects through the Name Service [COSS], which provides a directed graph of naming contexts with potentially many roots. A point in the graph may be reached via many routes, which means that an object may have many names. This is in contrast to OSI management where there is a naming tree instead of a naming graph and objects have exactly one name. The name server may be essentially used to assign names to objects and to resolve names to object references.

The example presented previously may be expressed in terms of CORBA objects in a one-to-one mapping with the equivalent OSI managed objects. A key difference is that there is no need for a containment tree as such but containment may be treated as any other relationship. Despite that, It will probably be necessary to model containment in order to assign unique names to managed objects, in a similar fashion to OSI management. Those objects may not be "discovered" and selected based on containment relationships, as is the case in OSI management, but through the trader. Discovery and access aspects are addressed in the next section.

Finally, while polymorphism is a general property of object-oriented systems and, as such, is supported in CORBA, there is no notion of allomorphism. The latter may be supported by passing the interface type explicitly as an argument to operations. In this case though allomorphism will not be transparent as it is in OSI-SM. In addition, there is no built-in support for the discovery of the allomorphic interface types that an object supports through a facility similar to the OSI-SM *allomorphs* attribute.

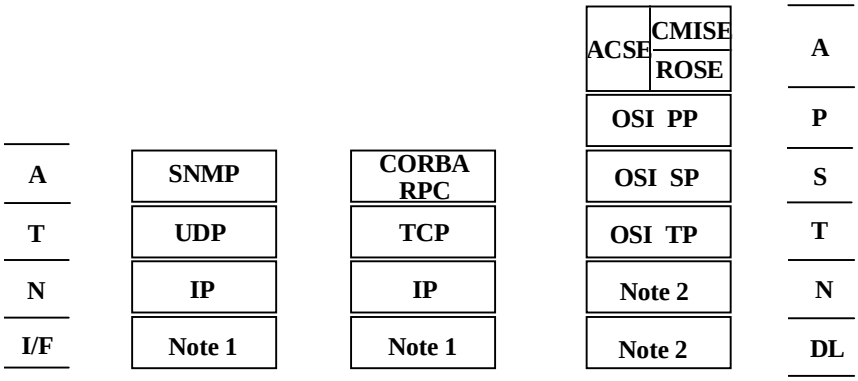
2.3. Access and Distribution Paradigm

In the three management frameworks, managing functions or objects implement management policies by accessing managed objects. By access paradigm, we mean the access and communication aspects between managing and managed objects. Access aspects include both the remote execution of operations on managed objects and the dissemination of notifications emitted by them. Given the different origins of OSI / Internet management and ODP / OMG CORBA, that is communications and distributed software systems respectively, there are marked differences in the relevant access paradigms. OSI and Internet management follow a protocol-based approach, with message-passing protocols modeling operations on managed objects across a management interface. The operations and parameters supported by those protocols are a superset of those available at the managed object boundary, with the additional features supporting managed object discovery and multiple object access. The protocol operations are addressed essentially to the agent administering the managed objects which acts as a naming, discovery, access and notification server. On the other hand, OMG CORBA specifies the API to the ORB through which client objects may perform operations on server objects. Remote operations are supported by a Remote Procedure Call (RPC) protocol. The latter carries the remote operation parameters and results, while functions such as object discovery and multiple object access are left to application services such as naming and trading.

While both OSI and Internet management have tried to optimize access aspects with respect to the target management environments, they have paid less attention to distribution aspects. By distribution, we mean the way in which managing and managed systems and objects discover each other and how various related transparencies, such as location, are supported. In OSI management, distribution has been recently addressed through discovery and shared management knowledge services [X750], supported by the OSI Directory [X500]. In the SNMP world, distribution is partly addressed through predefined addresses. On the other hand, OMG CORBA is influenced by ODP [X901] and, as such, from the beginning it has been designed with distribution and various transparencies in mind. Its ORB-based architecture has targeted the optimal provision of distribution, in the same fashion that the manager-agent architecture adopted by OSI and Internet management has targeted the optimal support for managed object access services. In this section, we look at and compare the access and distribution aspects of the three frameworks.

2.3.1 The Internet SNMP

In the Internet management framework, the fundamental axiom has been simplicity at the agent part of the manager-agent spectrum. This important design decision aimed at the provision of standard management interfaces to the majority of network devices at a minimal cost and has influenced the associated access paradigm. SNMP has been designed to support a connectionless style of management communication and, as such, it has been mapped over the User Datagram Protocol (UDP) and the Internet Protocol (IP) as shown in Figure 2.6 [RFC1157][RFC1905/6]. The relevant thinking is that reliable transport protocols such as the Transport Control Protocol (TCP) impose too much memory and processing overhead to be supported by simple network devices such as routers and bridges. In addition, maintaining transport connections requires state information, which again is considered to be expensive and, as such, undesirable. Also, bearing in mind that SNMP projects a largely centralized model, it is impossible to maintain simultaneously thousands of connections from a Management Operations Center (MOC) to the managed devices.



Note 1: The Internet interface layer is deliberately left undefined
Note 2: There exist many OSI Network/DataLink protocol combinations

Figure 2.6 SNMP, CORBA and OSI Protocol Stacks

Applications using the SNMP protocol will need to ensure the reliability of underlying communication by undertaking retransmission. In other words, applications should try to emulate the functionality of a reliable transport protocol by setting timers every time they perform a transaction and possibly retransmit. An important related aspect is that only applications in manager roles do retransmit, agents simply respond to requests. As such, there is the possibility that either the request or the response packet is lost. If the response packet is lost, the management operation will eventually be performed more than once; this is fine for information retrievals but might cause problems for intrusive operations. As such, either the latter should be *idempotent* or measures should be taken in the agent to prevent an operation from being performed twice (test-and-set, etc.). Finally, agents in managed devices do not retransmit notifications (*traps*). It is only applications in dual manager-agent roles that are allowed to retransmit notifications (*inform-requests*) in SNMPv2.

The management protocol operations are a superset of the operations available at the managed object boundary. We have already mentioned that SNMP objects accept only Get and Set operations and that imperative commands (actions) and table entry creation and deletion are emulated through Set. As such, the SNMP protocol also has *Get* and *Set* request packets. Agents emit notifications, which are sent to manager applications through the *Trap* SNMP packet. In addition to those operations, it is necessary for manager applications to be able to “discover” transient objects, for example table entries for which it is impossible to know their names in advance, e.g. connections, routes, etc. We have already mentioned before that the SNMP naming architecture relies on ASN.1 object identifiers. Since the latter have ordering qualities, a collection of SNMP objects visible across an interface is ordered in a linear fashion which means there is a first and a last object. This linear structure was depicted in Figure 2.4. Note that table entries are ordered on a column-by-column basis which is why they are depicted horizontally in that figure. It is exactly this linear structure that is exploited through the *Get-next* operation which allows to retrieve the next object of any other object in the

MIB. This is typically used for traversing tables and allows the retrieval of one table entry at a time if the entry names are not known in advance.

In SNMPv2, two more primitives have been added. The *inform-request* packet is to be used between hybrid manager-agent applications in order to report asynchronous notifications to each other; its receipt should be acknowledged by the receiving application. This means that it should be retransmitted, so that it can be thought as a "reliable trap". Note however that managed elements are not supposed to use this facility. The second addition has been the *Get-bulk* primitive; which is an extension of *Get-next* as it allows retrieval of more than one next objects. Although it is better than *Get-next*, it is still a poor bulk data retrieval facility because SNMP does not allow multiple replies but the result should fit in one response packet. Given the fact that the underlying transport is unreliable, the maximum allowed application-level packet size is about 500 bytes in order to avoid network-level segmentation. In short, the SNMP mode of operation is request-reply or request only for traps. The possible interactions using SNMP operations are shown in Figure 2.7.

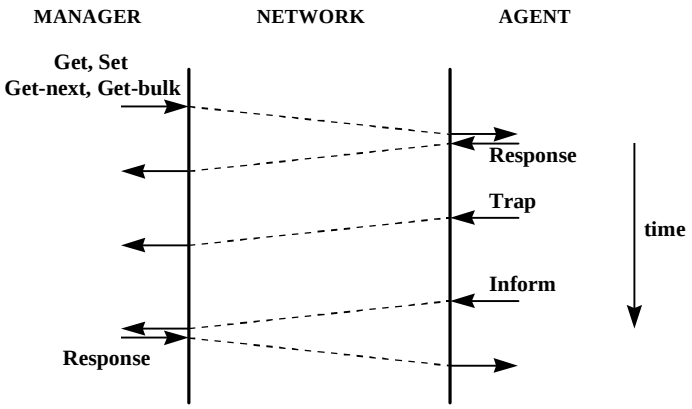


Figure 2.7 SNMP Protocol Interactions

Event-based operation in SNMP is pretty simple-minded. Traps are sent asynchronously from managed devices to predefined managers and are not retransmitted. This means they are inherently unreliable; as such, managers should not rely on them but should also monitor managed devices through periodic polling. This approach does not scale as it imposes management traffic on the managed network even when nothing happens. It also requires careful trade-off between the polling frequency and the potential elapsed time after a significant change before a manager knows about it. In SNMPv2, the event model between hybrid manager-agent applications is more flexible as it allows the request of particular events by creating entries in a relevant table. These events are retransmitted until an acknowledgment is received. On the other hand, it is not possible to specify any other event-associated conditions through filtering, in order to minimize further management traffic.

Finally, distribution in terms of location transparency is not an issue in SNMP. Manager applications address agents in managed devices through their IP address while the agent is always attached to the UDP port number 161. This simplistic model allows for the discovery of device agents in LANs and MANs because of their broadcast nature: a multicast SNMP message is sent to all the nodes of the LAN at the known port number, requesting the first MIB object, for example *Get-next(name=0.0)*, and the nodes that respond have been "discovered". On the other hand, such discovery is not possible for higher-level hybrid manager-agent applications as it is not specified on which port they should be attached. The only aspect that is specified regarding manager applications is that they should be listening on UDP port 162 for traps and confirmed inform-requests. In summary, the SNMP distribution model is very simple, in a similar fashion to the whole framework. It serves well enough the centralized management model of LANs and MANs but it has obvious limitations in more complex environments.

We will now consider a concrete example of SNMP usage in order to demonstrate the access framework. Assume we would like to find all the routes in the routing table of a network element that "point" to a particular next hop address. The manager application must know the logical name of that network element which it will map to the network address, typically by using the domain name system. It will then send a SNMP *Get-next* request to port 161 at that address, starting with an "unnamed" route table entry, which will result in the first route entry being returned. This request may be formed as *Get-next(routeDest, nextHopAddr, routeMetric)*. It will then have to repeat this step, passing each time as argument the result of the previous request. Having retrieved the whole table, the manager will have to filter out unnecessary entries and keep those for which the *nextHopAddress* object has the desired value. In SNMPv2, more than one next entries may be requested through the *Get-bulk* primitive but it might not be possible for the responding agent to fit all of them in the maximum SNMP packet size. In either case, the whole table needs to be retrieved in order to find particular entries, which is expensive in terms of management traffic. In addition, Get-next or Get-bulk requests need as argument the result of the previous request when traversing "unknown" tables. This means the overall latency for performing this operation will be a multiple of the latency incurred for a single retrieval.

Assume also that we would like to be informed if such a new route is added to the routing table, either by management or through a routing protocol. Since traps are used sparsely in SNMP and are also unreliable, the only way to discover new entries is to retrieve periodically the whole routing table. It should be added though that after the table has been retrieved once, names of existing entries are known and next traversals can start simultaneously at various "entry" points, reducing the overall latency. Despite that, the management traffic incurred will be roughly the same.

2.3.2 OSI System Management

OSI-SM was designed with generality in mind and as such it uses a connection-oriented reliable transport. The relevant management service/protocol (CMIS/P) [X710/11] operates over a full seven layer OSI stack using the reliable OSI transport service. The latter can be provided over a variety of transport and network protocol combinations, including the Internet TCP/IP using the RFC1006 method. The CMIP protocol stack is depicted in Figure 2.6. End-to-end interoperability over networks with different combinations of data link and network layer protocols is supported either through network-level relaying or transport-level bridging as specified in [Q811/12]. The upper layer part is always the same and comprises the OSI session and presentation protocols with the and Association Control Service Element (ACSE) and the CMIS Element over the Remote Operation Service Element (ROSE) in the application layer [Q811/12]. The benefit of transport reliability is out-weighted by the fact that a full seven layer infrastructure is required even at devices such as routers, switches and multiplexors, which typically run only lower layer protocols. In addition, application level associations need to be established and maintained prior to management operations and the reporting of notifications.

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

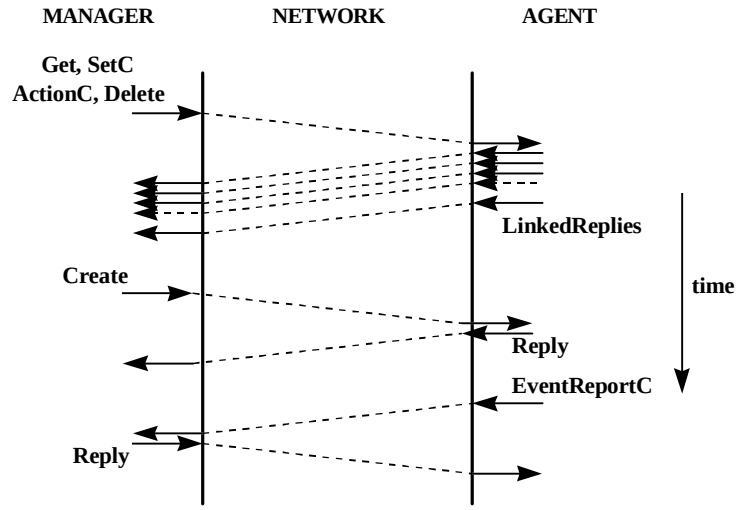


Figure 2.8 CMIS Interactions

When applying an operation to multiple objects through scoping and filtering, atomicity may be requested through a *synchronization* parameter. The result/error for each managed object is passed back in a separate packet, which results in a series of *linked replies* and an empty terminator packet. A manager application may interrupt a series of linked replies through the Cancel-get facility. Finally, the Set, Action, Delete and Event-report operations may also be performed in an unconfirmed fashion. While this is typical for event reports (the underlying transport will guarantee their delivery in most cases), it is not so common for intrusive operations as the manager will not know if they succeeded or failed. Nevertheless, such a facility is provided and might be used when the network is congested or when the manager is not interested in the results/errors of the operation. Figure 2.8 depicts the interactions between applications in manager and agent roles using CMIS (apart from Cancel-get).

The event reporting model in OSI management is very sophisticated, allowing fine control of emitted notifications. Special support objects known as Event Forwarding Discriminators (EFDs) [X734/5] can be created and manipulated in agent applications in order to control the level of event reporting. EFDs contain the identity of the manager(s) who wants to receive notifications prescribed through a filter attribute. The filter may contain assertions on the type of the event, the class and name of the managed object that emitted it, the time it was emitted and other notification-specific attributes, for example, for an attributeValueChange notification, the attribute that changed and its new and old values. In addition, an emitted notification may be logged locally by being converted to a specific log record. The latter is contained in a log object created by a manager, which contains a filter attribute to control the level of logging. In summary, OSI management provides powerful mechanisms for dealing with asynchronous notifications and substantially reduces the need for polling. In that respect, it scales much better than SNMP.

Distribution aspects in OSI management are supported by the OSI Directory, which provides a federated hierarchical object-oriented database. The Directory resides in many Directory Service Agents (DSAs) that administer parts of the global Directory Information Tree (DIT). Parts of the global MIT belong to different Autonomous Administrative Areas (AAAs) and start at Autonomous Administrative Points (AAPs). DSAs are accessed by applications in Directory User Agent (DUA) roles via the Directory Access Protocol (DAP) while DSAs communicate with each other via the Directory System Protocol (DSP). Accessing the local DSA is enough to search for information anywhere in the global DIT. Figure 2.9 depicts the operational model of the directory and the global DIT. Directory Objects (DOs) are named using distinguished names that express containment relationships, in the same fashion as OSI managed objects. In fact, the directory naming architecture preceded that of OSI management and was essentially reused in the latter.

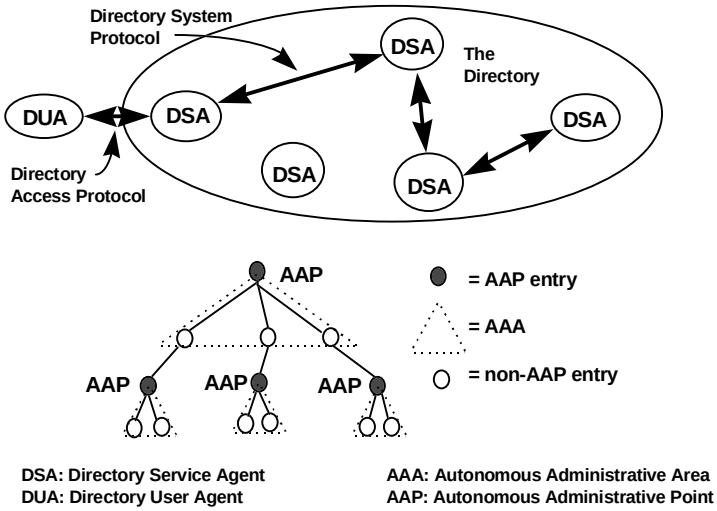


Figure 2.9 X.500 Directory Organizational and Administrative Model

OSI management applications or, System Management Application Processes (SMAPs) in OSI parlance, are represented by directory objects. The latter contain System Management Application Entity (SMAE) objects associated with each interface of that SMAP. SMAE DOs contain addressing information as well as information regarding other aspects of that interface, termed Shared Management Knowledge (SMK) [X750]. Since the same hierarchical naming architecture is used for both the OSI directory and management, the two name spaces can be unified. This can be achieved by considering a "logical" link between the topmost MIT object of an agent and the corresponding SMAP directory object.

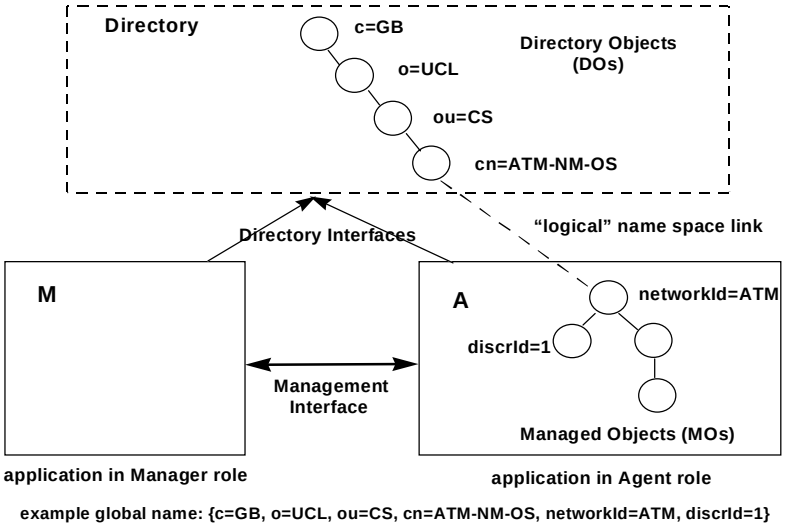


Figure 2.10 The OSI Global Name Space

The universal name space is shown in Figure 2.10 through the extended manager-agent model. The manager application may address objects through their global names, starting from the root of the directory tree, for example {c=GB, o=UCL, ou=CS, cn=ATM-NM-OS, networkId=ATM, logId=1, logRecordId=5}. The underlying infrastructure will identify the directory portion of the name, that is {c=GB, o=UCL, ou=CS, cn=ATM-NM-OS}, will locate the relevant DO and will retrieve attributes of the contained SMAE DO, including the OSI presentation address of the relevant interface. It will then connect to that interface and access the required managed object through its local name, that is {logId=1, logRecordId=5}. Note that the networkId=ATM relative name of the topmost MIT object is not a part of the local name. Global names

guarantee location transparency as they remain the same even if the application moves: only the presentation address attribute of the relevant SMAE DO needs to change. Note finally that applications in manager roles are also addressed through directory distinguished names regarding the forwarding of event reports since the destination address in EFDs contains the directory name of the relevant manager.

We will now consider the same example we considered in SNMP in order to see in practice the use of the OSI management access facilities. In this case the manager application will know the logical name of the device, for example {c=GB, o=UCL, ou=CS, cn=router-A}, from which the presentation address can be found through the directory. The directory access protocol offers facilities similar to CMIS scoping and filtering. A request will be sent to the SMAP DO with that name and the psapAddress attribute of the contained SMAE DO will be requested. The manager will then connect to that address and request the relevant table entries through scoping and filtering in the following fashion: *Get(objName={subsystemId=nw,protEntityId=clnp,tableId=route}, scope=1stLevel, filter=(nextHopAddr=X), attrIdList={routeDest, routeMetric}).* The results will be returned in a series of linked replies, sent back-to-back as shown in Figure 2.8. The overall CMIS traffic will be kept fairly low: N linked replies for the matching entries together with the request and the final linked reply terminator packets, that is N+2 in total. The overall latency will be slightly bigger than that of a single retrieval. It should be added that connection establishment and release are necessary both to the local DSA and the element agent. This does not happen on a per management request basis, but connections may be "cached", as already explained. It should be noted that the discovery of the element agent address through the directory is necessary only once.

The manager application would also like to be informed about new route entries "pointing" to the next hop address X. This could be done by using the rich event reporting facilities provided by OSI management. The manager will have to create an EFD with filter *(eventType=objectCreation AND objectClass=routeEntry AND nextHopAddr=X)* and set as destination its own logical name, for example {c=GB, o=UCL, ou=CS, cn=mgr-Z}. After that, notifications will be discriminated locally within the agent and the ones matching the filter will be forwarded to the manager. Note that if there is no connection to the manager, the element agent will have to establish it by going through the same procedure and mapping the logical manager name to an address through the directory. The previous observations about connection caching and address mappings are also valid in this case.

2.3.3 OMG CORBA

OMG CORBA was designed as a distributed software infrastructure in which the access protocol is secondary compared to the underlying APIs or "programming language bindings". Of course, an agreed protocol is necessary in order to achieve interoperability between products of different vendors. The OMG 1.x versions of CORBA specification left completely open the choice of access protocol and concentrated only on concrete programming language bindings. Version 2.0 also specified a Remote Procedure Call (RPC) protocol as the General Inter-Operability Protocol (GIOP) [GIOP]. Two different transport mappings have been defined for the latter, the Internet Inter-Operability Protocol (IIOP) [IIOP] over the Internet TCP/IP (shown in Figure 2.6) and the DCE Common Inter-Operability Protocol (D-CIOP). The ODP *access* transparency prescribes independence of the underlying access protocol and CORBA provides both independence and portability due to the agreed APIs. The access protocol could change without any effect on application-level software!

The agreed CORBA protocol is a connection-oriented reliable RPC that uses TCP and IP as transport and network protocols respectively. Applications that use CORBA-based communications are guaranteed transport reliability, in a similar fashion to OSI management and unlike SNMP. The CORBA RPC protocol is a *request/response* type of protocol in which the exact structure of the request and response packets is defined by the IDL specification of the accessed CORBA interface. No special facilities are built in the protocol for object discovery and multiple object access in a similar fashion to the SNMP get-next, get-bulk or the OSI management scoping and filtering. Instead, such facilities are provided in a limited fashion by the ORB and by special *servers*. In summary, the CORBA RPC protocol provides a *single* object access mechanism with higher-level facilities provided by standard OMG servers [COSS].

Chapter 2

The CORBA operational paradigm is different from that of OSI and Internet management, as it originates from the distributed system world. CORBA objects are specified and accessed separately, in contrast to the managed object cluster administered by an agent. Another key difference is that CORBA objects are most commonly addressed by *type* and not by *name*. This is due to the nature of distributed systems where, typically, instances of the same type offer exactly the same service, for example printer servers, statistical calculation servers, and so on. Of course, this does not mean that there are no support mechanisms to distinguish between instances of the same type (name servers, traders). It means however that the whole framework is optimized towards a "s*ingle object access, address by type*" style of operation, in contrast to the manager-agent model which is optimized for "*multiple object access, address by name*" style of operation.

A CORBA object instance can be addressed by type through the ORB, in a fully location-transparent manner. The ORB will find an instance of that type and return an object reference to the client object. If there are many instances of that type in the ORB domain, many references will be returned. Instances of the same type can be distinguished through naming servers or traders. A naming server [COSS] can be used to map a name to an interface reference. When an object instance is created, the naming server needs to be "told" of the mapping between the object's name and its interface reference. Subsequently, client objects can resolve object names to object references through the naming server. We shall recall here that the OMG naming architecture is very similar to that of OSI management / directory, but objects can have more than one names.

A trader [X9tr] supports more sophisticated queries, matching sought properties of the target object(s). Objects can export their interfaces to the trader together with a list of *attributes* and a list of *properties*. Clients may request the object references of a particular type that match assertions on attributes and properties. The difference between the latter is that attributes may change dynamically while properties are fixed during the lifetime of an object instance. As such, the trader needs to evaluate assertions on attributes by retrieving them from all the instances of the type associated with the query. The function of the trader is very similar to filtering in OSI management. A key difference is that only interfaces of a particular type can be searched through the trader. An additional difference is that filtering is tightly coupled with OSI managed objects through the supporting agent while the ODP/OMG trader is a separate server. Finally, traders can be in principle federated in order to cope with big object spaces and different administrative domains.

Notifications in ODP/OMG are supported by *event* servers. Emitting and recipient objects need to register with the event server and special objects called *channels* are created and managed for every type of notification. Emitting objects invoke an operation on the relevant event channel while the notification is passed to registered recipient objects either by invoking operations on them (*push* model) or through an operation invoked by the recipient object (*pull* model). There is no filtering as in OSI EFDs while event servers can be in principle federated for scalability and inter-domain operation. The key difference with OSI management is the lack of fine grain filtering which results in less power and expressiveness and more management traffic. OMG is currently working towards the specification of *notification* servers which will provide filtering and will also take over the management of channels, providing a higher-level way to deal with notifications.

We will now examine how CORBA could be used for network and service management, contrasting its approach to the protocol-based OSI and Internet management approaches. But let's first recapitulate the operational paradigm of the latter. Managed elements or management applications that assume an agent role provide management interfaces. A management interface consists of the formal specification of management information and of an access service/protocol that is mapped onto a well defined protocol stack. While the management information specification provides the MIB *schema*, object discovery and multiple object access facilities allow applications in manager roles to discover dynamically existing object instances. Operations to objects are always addressed through the supporting agent, which provides query facilities in a database-like fashion. In addition, the agent discriminates emitted notifications according to criteria preset by managers. Applications may discover each other through the directory in OSI management, while predefined addresses are used in SNMP.

If CORBA is used as the underlying access and distribution mechanism, managed objects can be mapped onto CORBA objects, accessed by client objects in managing roles. The key difference is that clusters of managed objects logically bound together, e.g. objects representing various aspects of a managed network element, are not seen collectively through an agent. As such, an important issue is to provide object discovery and selection facilities similar to OSI scoping and filtering. Such facilities are very important in management environments where many instances of the same object type typically exist, with names not known in advance, e.g. call objects. Facilities similar to scoping are not currently supported in CORBA but it should be possible to extend name servers to provide similar functionality since they maintain the logical name space. Facilities similar to OSI filtering are currently provided by traders, as explained above, but are not as powerful. An alternative solution would be to provide special *query* servers, offering object selection facilities based on scoping and filtering, in a similar fashion to OSI management.

The problem with the use of CORBA as described above is that federation is a key aspect in order to achieve scaleable systems. In essence, it will be necessary to have dedicated name servers, traders and event/notification servers for every logical cluster of managed objects, for example in every managed element, in order to reduce traffic and increase real-time response. Those "low-level" servers will be unified by "higher-level" servers in a hierarchical fashion but federation issues have not yet been worked out and are not simple. In addition, even with such facilities in place, the management traffic in terms of the required application-level packets will be at least double compared to that of OSI management. In CORBA, matching object references will be returned to the client object and the operations will be performed on an object-by-object basis. In OSI management, the multiple object access request will be sent in one packet while the results will be returned in linked replies, one for each object accessed. The use of CORBA for network management by using federated trading is depicted in Figure 2.11.

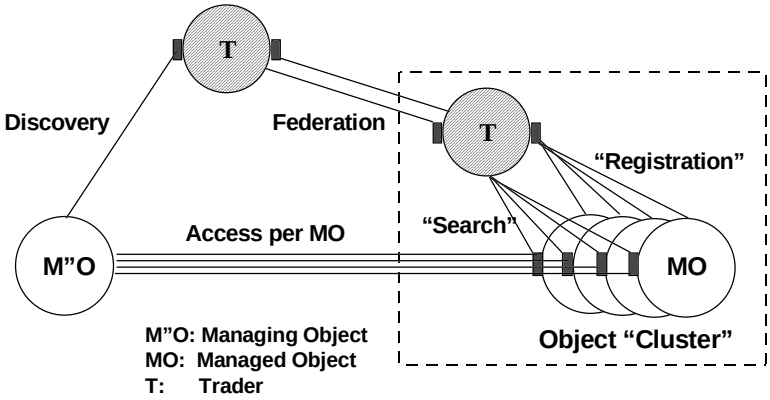


Figure 2.11 The Use of CORBA for Network Management

We will now consider the same example we considered in SNMP and OSI management in order to see in practice the CORBA access facilities. The manager in this case will be a CORBA client which will have to discover the right routing table entries through the trader. If there is no federation, a central trader will be used and objects such as route table entries will need to export their properties to it across the network. One of their properties will have to be the logical name of the router so that assertions about routes in different router nodes are possible. If there is federated trading, a trader could be located at the router node so that exporting properties by local objects would not generate any traffic. The manager will contact a trader and perform an operation to import interfaces with particular properties. The properties, in a filter-like notation, would be *(ifType=routeEntry AND router=router-A AND nextHopAddr=X).* This will result in a number of routeEntry interface references returned to the manager. The latter will then have to perform a get operation for each entry and retrieve the *routeDest* and *routeMetric* attributes. These operations may be performed concurrently, so the overall latency will be similar to that of one operation. Note, however, that the synchronous nature of RPC necessitates the use of a multi-threaded execution environment, with a separate thread for every invocation. The management traffic incurred will be 4 RPC packets for trading (2 to/from the domain trader and another 2

between the latter and the trader in the router); and 2*N RPC packets for retrieving the entry attributes i.e. 2*(N+2) packets in total.

Event operation in CORBA is less powerful than in OSI management but nevertheless useful for avoiding polling-based management. Assuming a *routeEntryCreation* event is defined, the manager will have to register with the event server in order to receive this event. Typically, every event will involve 4 RPC packets: 2 between the emitting object and the event server and 2 between the latter and the manager. Since OMG event services do not support filtering, the manager will receive events for all the new route entries in all the routers and will select locally those of interest. Federated notification servers will be necessary in the future to provide more sophisticated event management facilities.

2.3.4 Summary and comparison

In summary, SNMP adopts a connectionless unreliable transport while both OSI management and CORBA adopt a connection-oriented reliable transport paradigm. The only difference between the latter two is that connection establishment is "hidden" in the case of CORBA through the ORB while it may be presented to applications in the case of OSI management. The main reason for the connectionless (CL) approach in SNMP is simplicity in managed elements with complexity shifted to manager applications that have to achieve reliability through retransmission. Experience has shown that it is very difficult to optimize retransmission in the same fashion this is done by reliable transport protocols such as the Internet TCP and the ISO/ITU-T TP. In addition, compact implementations of reliable transport stacks have become a commodity while the recent advances in inexpensive memory and processing capabilities suggest that the argument of simplicity is no longer valid. Finally, the emerging broadband technologies (SDH/SONET transmission, ATM switching) are connection-oriented, which means they are better aligned with the OSI-SM and CORBA approaches.

Coming to the access paradigm, SNMP and OSI management adopt the manager-agent approach with managed object clusters visible across a management interface and query / event / multiple object access facilities while CORBA relies on a single object access paradigm, with special servers providing additional facilities. The query, event and multiple object access / bulk data transfer facilities of OSI management are very powerful as they have been designed specifically for telecommunication network management and have not been compromised in order to reduce agent complexity. The same facilities in SNMP are less powerful and they result in less expressive power and much more management traffic. On the other hand, they keep element agents simple and have resulted in making SNMPv1 a success, at least for private (LAN/MAN) networks. In OSI management and SNMP there are no implementation constraints in the sense that they are both communications frameworks. As such, compact optimized implementations are possible.

OMG CORBA, on the other hand, projects an object-oriented distributed software framework that is not specific to management and, as such, more general. Facilities similar to those provided by OSI/SNMP agents may be supported by OMG servers: name resolution and object selection will be supported by name servers, sophisticated filtering by traders while events are currently supported by event and, in the future, by more sophisticated notification servers. Given the fact that in management networks there will exist hundreds of thousands managed objects, federation is absolutely important for scalability and timely responses. The use of facilities such as name, event/notification servers and traders for network management is currently a research area while federation issues have not yet been resolved. An additional issue is the complexity of the overall resulting framework as CORBA dictates conformance to internal software interfaces, which leaves less space for optimized implementations. For example, the feasibility of network elements with tens of thousands of CORBA managed objects needs to be investigated.

2.4. Various other issues

2.4.1 Scalability, Flexibility

Scalability is an issue when managing large networks. Management services are used by the network operator and are transparent to end-users. Management traffic should be kept low so that most of the network bandwidth is available to end-user services. Obviously, the choice of management paradigm (event-driven, polling-based) has an impact on the amount of management traffic. We will examine a simple case study in order to quantify management traffic and assess scalability issues.

SNMP is particularly well-suited for LAN and MAN environments that are inherently connectionless, the available bandwidth is relatively high (10 to 100 Mbits/s) and the error rate is very low. In such networks, the overhead of polling is only a small fraction of the available bandwidth. No sophisticated retransmission is necessary because the small error rate allows communications to take place essentially over one "link". We will assume a modest LAN/MAN cluster consisting of 100 routers with an average of three interfaces per router and a maximum latency of five minutes for detecting "interface down" alarm conditions. This requires (100*3)/(5*60) = 1 polls/sec to monitor the up/down status of interfaces, or 1/3 polls/sec if all the requests for a router are combined in a single SNMP packet. Adding to this 5000 workstations/PCs/servers whose status needs to be known at a average latency of 10 minutes, we need another 5000/(5*60) = 8.33 polls/sec, the grand total being about 10 polls/sec. Since an SNMP packet is about 500 bytes, this results in a management bandwidth of 0.1 Mbits/s or 1 per cent of the total Ethernet bandwidth (10 Mbits/s), which is affordable.

The above calculations concern polling to determine only the rudimentary status of the network, in terms of its most important components. Adding to this system and application management, for example operating system load/users, terminal servers, database management systems, the domain name system or directories, mail systems, and so on, including performance and accounting issues in addition to fault detection and the above figure will be much bigger. It is obvious that doing the same thing over a wide area network with a lot of "thin" point-to-point links and higher probability of congestion will result in a lot of additional load, deteriorating the network's overall health. In such an environment, sophisticated retransmission mechanisms will be also necessary because the probability of packets being lost will be much higher. The solution for scalability is event-driven management, with facilities such as event management with filtering, event logging, metric monitoring with thresholding and summarization (see section 2.4.2). SNMP does not provide such facilities, at least for element management, while OSI management does. OMG CORBA on the other hand was not designed specifically for management, its event model is not as powerful and lacks generic management facilities.

We will now examine suitability for hierarchical management. By hierarchical management we mean a management system organization in which management applications are organized in a logical layered fashion, with applications in higher layers being shielded from unnecessary detail and having a global view of the network, services or policies. A hierarchical management structure was depicted in Figure 2.2 and is exemplified by the TMN model. The SNMP framework was designed to allow management capabilities to be fielded in the largest possible number of network elements. As such, simplicity dictated the use of connectionless transport, no sophisticated event facilities and a rather crude information model. Such design decisions address mainly the lowest level of a management hierarchy (element management). However, when it comes to hierarchical management, issues such as management application size, complexity and processing requirements become largely irrelevant. Management applications in this layered hierarchy usually operate in powerful workstations. In this case, the simplicity of the SNMP framework becomes a liability as it restricts the available expressive power and introduces limitations.

Finally, we will consider flexibility and suitability for distributed application and service management. Both SNMP and OSI management may be used for distributed application management. On the other hand, distributed applications should be managed through dedicated agents and both SNMP and OSI agents are too complex to be "bundled" together with them. In general, it is natural to manage distributed applications employing the same technology used to build them in order to achieve reusability and economies of scale.

Internet applications can be managed with SNMP, OSI applications can be managed with OSI and CORBA applications are best managed through CORBA. Given the fact that CORBA was conceived as a mechanism to build distributed systems, it is best to build and manage new distributed applications through CORBA.

The same is true for service management regarding new advanced services, for example video-conferencing and joint document editing. In this case, it is difficult to differentiate between service operation and service management. For example, subscription management to a new advanced service is a management activity that is closely related to the operation of that service. This is exactly the thinking behind the adoption of CORBA as the basis for the TINA DPE, as the TINA architecture tries to unify service operation and service management mechanisms and procedures. In this unified model, a video-conferencing bridge can be seen as a CORBA object with both service and management interfaces as opposed, say, to an object with a service interface and an associated OSI agent for TMN-based service management. The two approaches are depicted in Figure 2.12. In summary, OMG CORBA is a more flexible mechanism than SNMP and OSI management for *managing* distributed applications because it is also a mechanism for *building* them in the first place.

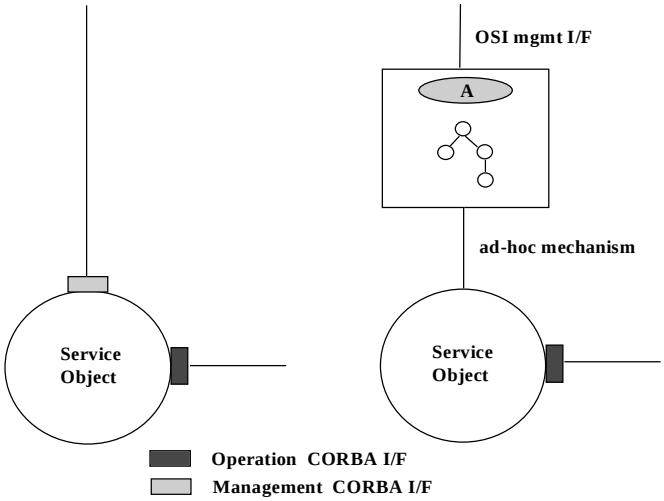


Figure 2.12 Service Operation and Management Models

2.4.2 Generic Management Functionality

One fundamental difference between the Internet and OSI management frameworks is that the Internet follows a "lowest common denominator" approach, resulting in very few common object specifications that should be globally supported. This approach is in line with its fundamental axiom which dictates simplicity in managed elements. On the other hand, a number of generic management functions are standardized in OSI management in order to provide a well-defined framework for dealing with common tasks and achieving reusability. These specifications emanate from the five functional areas (Fault, Configuration, Accounting, Performance, Security - FCAPS) and are collectively known as the System Management Functions (SMFs) [SMF]. The notion of generic functionality in CORBA is supported by the Common Object Services [COSS]. As CORBA was not designed specifically for management, it only partly supports functionality similar to the OSI SMFs. In this section, we examine generic management functionality through the OSI SMFs and compare it to similar facilities in the SNMP and CORBA frameworks.

There are three types of OSI SMFs:

i. those that provide *generic* definitions of object classes or simply attributes, actions and notifications for common tasks;
ii. those that provide *system* definitions which complement the management access service by providing a controlled mechanism to deal with notifications (Event Reporting / Dissemination, Log Control); and

iii. those that provide *miscellaneous* definitions; we could currently group here the security-related functions (Access Control Objects, Security Alarm Reporting, and Security Audit Trail).

Starting first from the third category, such functions exist in the SNMPv2 security framework (apart from security audit trail). The SNMPv2 Party MIB has similar functionality to the OSI Access Control Objects. These facilities are of paramount importance for the security of management and are absent in SNMPv1. The OMG CORBA security framework supports access control and security audit functions (see section 4.3).

The second category is extremely important as it provides the means for event-driven instead of polling-based management. The philosophy of both SNMPv1 and v2 is based on polling, at least between element managers and NEs. As such, similar facilities exist only partially, as already discussed. The event group of the "manager-to-manager" MIB provides facilities similar to OSI event reporting but without filtering while the same is true of the OMG event service. Logging services are not provided in either the SNMP or OMG CORBA frameworks.

The first category provides a host of functions that support generic functionality. The first and most important of those are Object Management, State Management and Alarm Reporting. Object Management provides three generic notifications related to configuration management that all OSI managed objects should support: *object creation*, *object deletion* and *attribute value change*. State Management provides a number of generic *state* attributes (administrative, operational, usage state, etc.) and a *state change* notification. It also prescribes state transition tables according to the state model. Finally, alarm reporting provides a set of generic *alarm* notifications: quality of service, communications, equipment, environmental and processing error alarm.

Other MIB specifications should use the above definitions in order to model object, state and alarm aspects. Generic configuration, state or alarm managers can be written in a fashion that makes them independent from the semantics of a particular MIB. For example, a configuration monitor could be an application that connects to managed elements and requests all the object creation, deletion, attribute value and state change notifications in order to display changes to the human manager. Such an application can be written once and reused as it only needs to be "fed" the formal specification of the element MIBs in order to be able to display meaningful names for the objects emitting those notifications. OSI management platforms typically provide a set of generic applications that are based on those common specifications. SNMP and CORBA do not provide similar generic facilities but CORBA may reuse the OSI ones if the relevant GDMO specifications are translated to CORBA IDL as described in the next section. This observation also holds for the rest of the OSI SMFs which are described next.

Monitor Metric objects allow the observation of counter and gauge attributes of other MOs and their potential conversion to *derived gauges*, which may be statistically smoothed. The latter have associated threshold and tidemark attributes that fully support event-driven performance management capabilities, relegating "polling" within a managed element. The SNMPv2 manager-to-manager MIB offers a similar facility but without statistical smoothing or the possibility of combining different attributes in order to produce a comparison rate, for example for error versus correct packets. Summarization objects allow a manager to request a number of attributes from different objects of a remote system to be reported periodically, possibly after some statistical smoothing. These attributes can be specified using scoping and filtering while intermediate observations may be "buffered". This facility is important for gathering performance data for capacity planning and is typically used together with logging. SNMP does not provide such a facility.

Accounting Metering provides generic objects to support data collection for resource utilization. Test Management defines generic test objects to provide both synchronous and asynchronous test facilities, modeling generic aspects of testing and separating them from specific test aspects. Scheduling Management provides generic scheduler objects that could schedule activities of other MOs which support such scheduling on a daily, weekly, monthly or other periodic basis, for example event forwarding discriminators and logs. Response Time Monitoring supports performance management by allowing the measurement of protocol processing time and network latency between systems. Time Management permits the delivery of correct time and synchronization of the clocks of distributed systems. Software Management allows the delivery,

installation, (de-)activation, removal and archiving of software in a distributed fashion. SNMP does not provide similar generic facilities.

In summary, SMFs provide useful generic facilities that most systems should support, enforcing a common style of operation that can result in generic managing applications or simply generic managing functions. SNMPv2 matches partly the event reporting and metric objects, and these only in the manager-to-manager domain. OMG CORBA provides less powerful event reporting facilities while the rest of the OSI SMFs could be translated to CORBA IDL and used in CORBA environments. Despite this theoretical possibility, this approach has not yet been put into practice.

2.4.3 Security

Security of management is of paramount importance, especially for *intrusive* operations that result in the modification of management information. Security is particularly important across different administrative domains but is also necessary within a domain, especially in cases where that domain is open to external management traffic. Despite its importance, it has taken a long time to produce agreed-upon workable solutions. A common aspect in all these frameworks is that security mechanisms have been almost an after-thought, after the main aspects have been standardized and non-secure implementations have existed in the market place for some time.

First, we will describe security threats and then security services used to protect against those threats. A third-party application may attempt to subvert the management interaction between a pair of communicating management applications by:

- **Masquerading** as a legitimate application and then performing unauthorized management operations;
- **Modifying** information while in transit;
- **Re-ordering** or **re-playing** messages in transit; and
- **Capturing** (and examining) confidential management information in transit.

The security services used to protect against these threats are the following:

- **Peer entity authentication** which establishes unambiguously the identity of the initiator of an operation and is an essential input to an access control decision function;
- **Data origin authentication** which provides an assurance guarantee that data really do come from where they seem to;
- **Connectionless integrity** which ensures that management PDUs cannot be modified without detection;
- **Stream integrity** which guards against mis-ordering PDUs in a stream (including re-plays);
- **Confidentiality** which prevents capture and examination of management information; and
- **Access control** which enables one to discriminate between different managers or client objects regarding the operations they are allowed on managed objects.

We will now consider the approaches taken for providing these services in the three management technologies. In the Internet management world, SNMPv1 has little security in the form of “password-based” authentication and access control [RFC1157]. There is a special field in SNMPv1 packets that identifies a *community* name. Every agent needs to know in advance the names of various communities with different access rights. As such, MIB access is restricted based on that name. The default community is called *public* and provides a minimal level of access, for example *read-only* for objects that can be accessed by anybody. The first problem with this scheme is that it does not allow for the dynamic configuration of agents with respect to communities and access rights. The second and most important problem is that the community name is passed across unencrypted, which makes the scheme vulnerable to *capturing* attacks. Because of its simple-minded nature, the SNMPv1 security mechanism is not trusted and, effectively, not used. In the absence of strong security mechanisms, many SNMPv1 manageable devices do not implement Set operations in order to avoid the potentially disastrous effects of malicious attacks. Even worse, the various IETF groups involved in the

definition of new MIB specifications have refrained from allowing extensive intrusive management capabilities through Set operations. The absence of security has resulted in the use of SNMPv1 as a remote *monitoring* rather than a *management* framework.

The initial Internet SNMPv2 RFCs, published in 1993, included a security framework but the relevant IETF working group never reached agreement regarding the security aspects. As such, the new version of the SNMPv2 RFCs, published in 1996, has made the security framework optional and allows for the possibility of multiple security frameworks. The simple community-based scheme can also be used with SNMPv2. A more comprehensive security framework caters for the attacks mentioned above by providing comprehensive security services [RFC1909/10].

In SNMPv2, the relevant communicating entities are referred to as *parties*. Peer-entity, data-origin authentication and connectionless integrity are supported by a scheme based on the Message Digest 5 (MD5) algorithm. With MD5 authentication, each party is associated with a secret key, held securely at source and destination. The source party applies the MD5 algorithm to a combination of PDU and secret key in order to generate a security checksum that is appended to the PDU. The destination repeats the calculation to verify the source party's identity. The use of shared secret keys introduces a problem of key distribution. This can be addressed by secure key exchanges through SNMPv2 itself. However, this can only be done if it is certain the security of the key exchange channel itself has not been compromised. In addition, there is a bootstrap problem to solve; that is keys will need to be distributed *out-of-band* for the first time. Stream integrity may be supported by timestamps included in every message, assuming clocks are synchronized. An alternative novel approach has been proposed for SNMPv2 which uses time as perceived by the agent only and, as such, it avoids clock synchronization. Confidentiality is supported by encrypting portions of an SNMPv2 message using the Data Encryption Standard (DES). Finally, access control is provided through the *party* MIB which describes the access rights of different parties to objects in the agent's MIB.

In OSI, security services other than access control are applicable to all the application service elements and not just to CMISE. For example, the same authentication, integrity and confidentiality services can be used for management, directory access and file transfer. Authentication services were initially developed for the OSI directory but the need for a general security framework led to the Generic Upper Layer Security (GULS) ITU-T recommendations [GULS]. These have been completed in 1996 while specific lightweight profiles have also been produced by workshops to ease their acceptance and introduction into the market place. A key difference between the OSI and SNMP security frameworks is that OSI also defines mechanisms for asymmetric public-key cryptography in addition to symmetric secret-key based schemes. In asymmetric schemes, the source application needs to demonstrate knowledge of its own *secret* key. This requires a certain amount of infrastructure support in the form of *certification authorities* which vouch for the bindings between the directory name of that application and its *public* key. There is no key-distribution problem as each entity holds (securely) its own secret key while it "advertises" its public key to the directory. Asymmetric public key cryptography requires the RSA algorithm which can be computationally expensive when performed in software while implementations in hardware (*smartcards*) overcome this limitation. Directory access by a management application is necessary for both advertising its own public key and obtaining a peer entity's public key.

OSI GULS services include authentication, connectionless / stream integrity and confidentiality. The default digest and encryption algorithms used are MD5 and DES respectively but any other algorithm can be used after negotiation. Two lightweight profiles have been defined by the ANSI T1M1 and the IEEE Open Implementers Workshop (OIW), the *mini-* and *micro-GULS* respectively. These may operate using secret key authentication, while key distribution aspects are deliberately left unspecified. The key difference between the two profiles is that micro-GULS supports the encryption of a whole PDU only while mini-GULS supports the encryption of selected PDU fields, providing additional flexibility and increased efficiency. While GULS requires a presentation layer PDU transformation and, as such, protects all application layer services, an alternative approach is to provide security services based on ROSE PDU transformation [Bha96]. Such an approach is simpler since it does not require the modification of the OSI presentation layer. On the other hand, it can only protect ROSE-based application services. The ANSI T1M1 and IEEE OIW have also produced ROSE-based security service specifications as a simpler alternative to GULS. While using GULS or ROSE-

based security services, access control is provided through special managed objects that protect other *target* objects or even individual attributes, actions and notifications in a very flexible manner. Management of security facilities is provided through security alarm reporting and security audit trail functions [SMF].

OMG CORBA security services are a fairly recent addition to the overall framework. The OMG security framework is very broad and supports a plethora of possible security services and mechanisms. The current security specification defines APIs that provide access to security services supported by a number of different, potentially replaceable, security architectures and policies. The security services accessible through those APIs include authentication, protected message exchange, that is integrity and confidentiality, access control, security auditing and delegation of security rights to intermediate authorities [OMGSEC]. In fact, the CORBA security approach offers a "shopping-list-oriented" solution space to which implementations of underlying security architectures and policies can adhere. Accordingly, interoperability is not the focus of this high-level specification. In order to foster interoperability, however, a set of profiles specifying particular security services and relevant supporting mechanisms have been defined. A simple negotiation protocol has been defined in order to choose a particular profile between ORBs.

In summary, OSI management and OMG CORBA security solutions have been fully specified and secure implementations of relevant products are expected to appear in the market place soon. On the other hand, SNMPv2 has opted for an optional security framework whose adoption is questioned.

2.5. Interworking and Coexistence

In this section, we examine interworking and co-existence aspects for the three different technologies. We look first at interworking and co-existence between OSI and Internet management in the TMN context, where SNMP-capable network elements may need to by managed by TMN applications. We then look at interworking between OMG CORBA and SNMP in the TINA context, where network elements need to be managed by CORBA objects. Both of these cases are unidirectional in the sense that SNMP is only considered in the managed end of the spectrum. Finally, we examine interworking and co-existence between the OSI management and OMG CORBA in both directions: first in the OSI management to CORBA direction, which is necessary to manage CORBA-based distributed applications from a TMN environment; and then, in the CORBA to OSI management direction, which is needed to manage network elements with Q interfaces or to re-use existing TMN management services in a TINA environment.

2.5.1 OSI and Internet Management

Interworking between OSI management and SNMP is mostly necessary to manage SNMP-capable network elements in a TMN fashion. This is particularly common for ATM equipment, as relevant SNMP information models have been available for some time before the relevant TMN recommendation, resulting in a number of SNMP-capable ATM elements in the market place. In transmission technologies such as SDH, the situation has been the reverse with early Q-compliant available elements. In general though, it is expected that in the short to medium term i.e. for the next few years, it will be necessary to manage SNMP-capable elements from a TMN environment.

Interworking between CMIS/P and SNMP has been a subject that led to a lot of research trying to bridge the two worlds. Various solutions have been proposed, all of which can be classified into two broad categories:

i. integration in the manager end; and
ii. integration in the agent end of the manager-agent model.

The integration in the manager end means that one accepts the diversity in the supported technology by managed elements and tries to provide element management applications that understand the different underlying information models and access mechanisms. This approach is often referred to as *dual stack*

manager, since these applications will need to understand both the OSI and Internet management models. Such an approach has been envisaged by the X/Open Consortium in providing the XOM/XMP [XMP] Application Programming Interface (API), which provides uniform access to both CMIS and SNMP services.

Such an approach is suspect, however, because it is difficult to conceal which model the managing application deals with since both the underlying information models and access mechanisms have important differences. For example, the nature of objects and the naming schemes are different in the two models. This is also the case with respect to the supported communication and access paradigms. Of course, this does not mean that such an approach is not feasible but simply that integration cannot be seamless, increasing significantly the development effort and investment for dual manager applications. The dual stack manager approach is depicted in the left part of Figure 2.13.

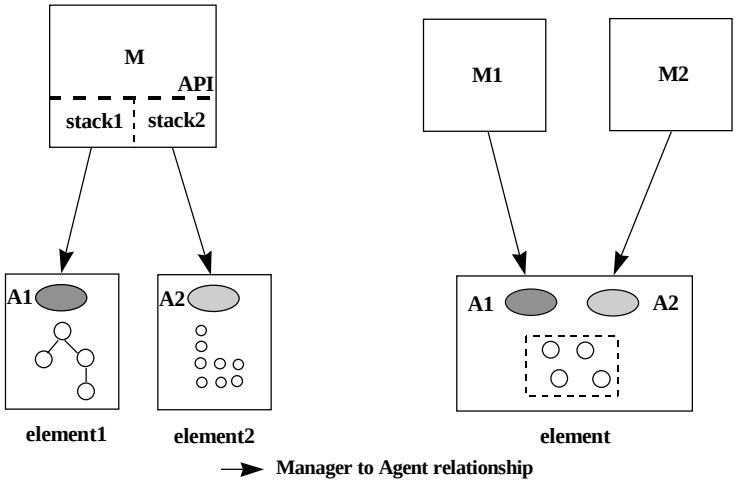


Figure 2.13 The Dual-stack Manager and Dual-stack Agent Approaches

Integration by agents can again be classified into two broad categories:

i. the dual agent approach; and
ii. the application gateway approach.

By *dual agent* we mean that two agents should exist for every managed element, both an SNMP and an OSI one. The information models will be semantically similar, which implies that the associated “real resource” aspects could be the same. As such, investment in providing such agents could be reduced if a modular approach were followed. In the latter, managed objects are realized in a “model/protocol independent” fashion and are associated with both SNMP and CMIS/P access methods. This approach is depicted in the right part of Figure 2.13, in which the objects in the dual agent are model independent, with different views presented through the two agents. Despite the fact that this approach is technically feasible, it requires heavy investment and additional resources in managed elements. As such, no products support this type of functionality to date.

The *application gateway* approach provides the most promising and powerful solution for integrating the two frameworks. In this, an application acts as a gateway (*proxy* or *adapter* are two other terms often used) for one or more agents of the other framework, exporting “converted” information models and providing service conversion from one access method to the other. The conversion between the information models can be performed either manually or automatically. *Manual conversion* means that human heuristics may be applied to result in an "elegant" model. In many cases, the target model for that technology may already exist, in which case a gateway should simply map one to the other. *Automatic conversion* means that a well-defined set of rules exists and can be used to automate translation of any MIB specification from one model to the other. As a result of automatic conversion rules, the dynamic interaction translation and subsequently application gateways may be automated.

Automatic conversion is usually unidirectional: it can only be bi-directional if the two information frameworks are equally powerful and expressive. This is not the case with OSI management and SNMP, GDMO being much more powerful than the SNMPv1/v2 SMI. As such, there may be automatic conversion of an SNMP information model to the equivalent GDMO one but not vice versa. Human intervention is required for mappings in the opposite direction. For example, there is no deterministic method for emulating a CMIS/P *Action* through an SNMP *Set*.

Research effort led by the Network Management Forum (NMF), known as the ISO/ITU-T and Internet Management Co-existence (IIMC) work, resulted in a set of NMF documents that provide rules for automating the SNMP to GDMO information model conversion and for building generic application gateways between CMIS/P and SNMP [IIMC]. The automatic conversion between the two frameworks relies on the simple observation that the SNMP structure of management information is a pure subset of the OSI one. SNMP objects are equivalent to OSI attributes, groups are mapped to classes and table entries become separate classes. Traps are mapped to notifications associated with a *cmipSnmpProxyAgent* class that represents the proxied SNMP element. Based on those rules, one can fully automate the CMIS/P to SNMP service conversion. Commercial products providing this functionality already exist. Typically, every time a new element with an "unknown" MIB needs to be adapted for, an off-line procedure is involved to let the gateway "know" of this MIB through suitable translators/compilers. The latter generate run-time support in a data-driven fashion so that the gateway logic does not need to be altered. The generic gateway is a Q-Adapter in TMN terms. The application gateway approach is shown in Figure 2.14.

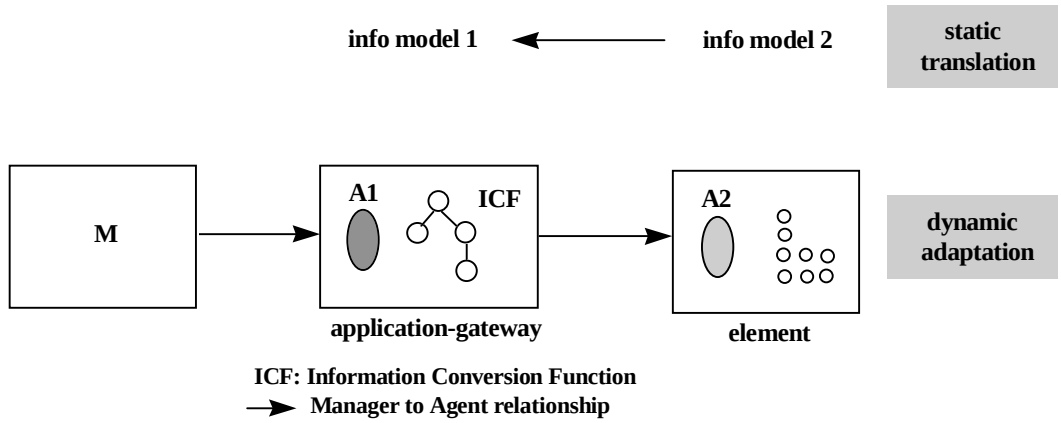


Figure 2.14 The Application-gateway approach

The important aspect of the generic gateway approach is that investment is rather small compared to the end result, which is OSI manageability of any SNMPv1/v2 capable element. The key benefit OSI management brings to the SNMP world is event-driven management through the Systems Management Functions. For example, metric and summarization functions together with event reporting and logging may be used to provide sophisticated management capabilities, eliminating polling in the local environment between the gateway and the proxied SNMP agent. The benefits of the generic gateway approach are described in [Pav95b]. The key drawback on the other hand is that the resulting information model does not exploit the object-oriented aspects of GDMO: inheritance is only two-level (every class inherits only from *top*)while containment is also fairly "flat". Furthermore, the resulting information model needs to be standardized in order to be considered a standard Q interface in TMN terms.

2.5.2 ODP/OMG CORBA and Internet Management

Interworking between OMG CORBA and SNMP is necessary in order to manage SNMP-capable network elements in a TINA environment. Early TINA prototypes have been using non-generic adapters supporting only the necessary functionality, e.g. for Connection Management. As other parts of the TINA management

architecture, such as Resource Configuration Management, are further specified and expanded, the need to access SNMP-capable elements will be much greater. If in the long term TINA plans to provide full scale TMN-like functionality, SNMP or Q-capable network elements will need to be accessed by CORBA managing objects. A generic approach to information model translation and dynamic adaptation will pay dividends as it will minimize the necessary investment and will allow the reuse of the relevant adapters. The need to interwork between OMG CORBA and both Internet and OSI management led to the joint effort between X/Open and the NMF, known as the X/Open Joint Inter-Domain Management (XoJIDM) task force [JIDM]. In this section, we concentrate on the issues behind OMG CORBA and SNMP interworking.

CORBA IDL is a more powerful object-oriented interface specification language than the SNMPv1/v2 SMI templates. In addition, the NMF IIMC work for mapping a SNMP SMI model to the equivalent GDMO one is of direct relevance and the same modeling principles apply for translation to CORBA IDL. SNMP objects can be mapped onto IDL interface attributes, groups can be mapped onto IDL interfaces and table entries can be mapped onto separate IDL interfaces. Finally, traps become notifications modeled by two IDL interfaces: a *Notification* interface that should be inherited by any managing object wishing to receive notifications according to the push event model; and a *NotificationPull* interface that should be inherited by notification server objects supporting the pull event model.

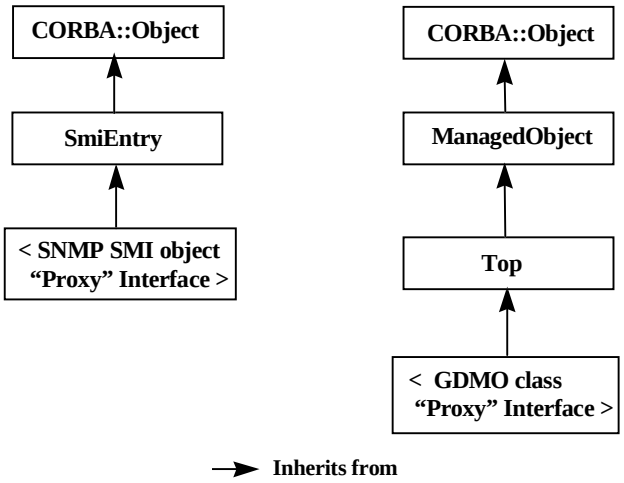


Figure 2.15 Inheritance Hierarchy from SNMP SMI and GDMO to IDL Translation

Every translated IDL interface inherits from a *SmiEntry* base interface, which in turn inherits from CORBA's *Object*, as do all IDL interfaces (see Figure 2.15). *SmiEntry* provides generic SNMP-related functionality in the form of a "naming" attribute and other generic aspects. While SNMP is not particularly powerful as an access method when compared to CMIS/P, it still offers some access facilities that cannot be easily provided by CORBA. For example, one SNMP request may retrieve or change the value of attributes across different table entry instances, for example the status of interfaces at a particular node and the next hop address of routes. In CORBA, objects are seen as different entities through IDL interfaces and, as such, a separate method invocation is needed for each interface.

2.5.3 ODP/OMG CORBA and OSI Management

Interworking and co-existence between OSI management and OMG CORBA is needed in both TMN and TINA environments: it should be possible to manage CORBA-based distributed applications from a TMN environment, e.g. in the context of service management; and it should be possible to access TMN-compliant elements from a TINA environment or to re-use existing TMN-based management services. Some observers see this latter case as a possibility for a TMN to TINA coexistence strategy: the TMN network layer management services could be re-used, with the TMN service layer being replaced by equivalent TINA functionality.

In order to manage CORBA server objects through OSI management we need to first translate IDL to GDMO/ASN.1 and then to provide mappings between the CMIS/P and the CORBA access mechanisms. Mapping CORBA IDL interface definitions to GDMO classes is fairly straightforward since IDL is simpler than GDMO. IDL attributes are mapped onto GDMO attributes, IDL methods are mapped to GDMO actions and IDL interfaces to GDMO classes. CORBA object references and names will be mapped onto OSI distinguished names. The generic application gateway needs to interact with standard OMG services in the CORBA domain, for example the OMG Name Service to resolve distinguished names to object references, the OMG Lifecycle Service to create new object instances and the OMG Event Service in order to receive events and forward them to interested OSI managing applications. Scoping and filtering can be resolved within the gateway, with one CMIS request mapped onto one or more requests on IDL interfaces. This type of gateway can also be conceived as an OSI agent for which the real resources associated with the managed objects it administers happen to be CORBA objects.

Mapping in the opposite direction is a more difficult proposition. GDMO/ASN.1 as an information specification language and CMIS/P as the access method have a number of aspects for which there exist no IDL and CORBA equivalents. These include the late binding of functionality to managed object instances through the use of conditional packages; the existence of notifications as part of managed object specifications; the fine grain support for event discrimination; and the use of scoping and filtering as "query language" facilities that may result in multiple replies. In addition, a GDMO action on a single managed object instance may also result in multiple replies, e.g. a testing action taking a long time to execute with periodic results. It should also be noted that GDMO attributes cannot be mapped directly onto IDL attributes since user exceptions with specific error information may be raised as a result of access to them. In IDL it is not possible to associate user exceptions with attribute access.

Despite these differences, it is still possible to use workarounds in order to achieve a generic mapping. GDMO attributes may be mapped onto access methods specific to the attribute in hand, according to its property information (e.g. *administrativeState_get*, *administrativeState_set*). GDMO actions resulting in single replies may be naturally mapped onto IDL methods. Actions resulting in multiple replies may generate exceptions to draw the attention of the calling object, with the replies modeled as methods in the opposite direction. Notifications may be mapped onto interfaces in the opposite direction, corresponding to the push and pull models. Finally, conditional packages can be made "mandatory" by being added to the resulting IDL interface. Their presence, however, becomes an implementation issue: the standard CORBA *not_implemented* exception should be raised whenever a method of a non-implemented package is invoked. Translated IDL interfaces follow exactly the same inheritance lattice as the original GDMO classes, while the *Top* class inherits from a *ManagedObject* base interface which in turn inherits from CORBA's *Object*, as do all IDL interfaces (see Figure 2.15).

The suggested mapping goes a long way towards reconciling the differences of the two object models but some semantics are inevitably lost in the translation. Most notably, in GDMO conditional packages may or may not be included in an object instance at creation time. This facility allows for the late binding of functionality to that instance and it may also be used to configure its "mode" of operation. This cannot be achieved through the suggested translation. Furthermore, some conditional packages for the same class may be mutually exclusive; this again cannot be modeled in IDL. If ISO and ITU-T are to adopt the proposed translation guidelines by XoJIDM, they should also instruct GDMO information modeling working groups to avoid the use of conditional packages in a non IDL-compatible fashion.

A more important difference concerning the translation has to do with the access methods. The operational model of CORBA is that of a single distributed object, accessed in a location transparent fashion. In OSI management, managed objects can be accessed collectively through the CMIS/P scoping and filtering facilities. These may be used for discovery services, for example "which calls are currently established through that element", and they minimize the management traffic incurred on the managed network. In addition, the same operation may be performed on many managed objects. This not only is an engineering-level optimization but also allows a higher level of abstraction to be provided to managing functions. Discovery facilities may be

provided through traders in CORBA as discussed in section 2.3.3 but the efficiency of such mechanisms, with potentially thousands of transient managed objects in network elements, needs to be evaluated. In addition, the CMIS/P operational paradigm with potentially multiple operations expressed through a single request is lost, unless similar facilities are provided through special CORBA servers, as discussed in section 2.6.

2.6. Summary and The Future

Here we summarize the key aspects of the three frameworks, make final comments on their suitability for telecommunications network / service management and look at possible future directions.

The Internet management framework was conceived mainly for LAN/MAN management. It is a communication framework based on the manager-agent model whose design decisions opted for agent simplicity, shifting sophistication and complexity to manager applications. It has adopted a connectionless unreliable transport mechanism, a rudimentary object-based information model and a polling-based model for element management. It follows a "lowest common denominator" approach to management standardization, addressing only the absolutely necessary aspects. Version 1 offers little security while managed object creation/deletion is problematic. Its simplicity for managed devices has made it successful in the Internet network element market. Version 2 has only recently been completed. It fixes some of the problems of version 1 (e.g. object creation/deletion), but its adoption is questioned given the overall cost of the transition compared to the new features. In addition, the security framework has not yet been fully agreed and it is optional, which implies that the overall framework will continue to be used for mostly monitoring rather than intrusive management.

Telecommunications environments guarantee quality of service. They need to support a very high degree of availability and fault-free operation and are inherently connection-oriented. Internet management does not well match requirements such as timely reaction to network events, minimization of management traffic, geographic dispersion of control through distribution and strong security guarantees. In addition, the simplicity of agents is not a big issue for telecommunications network elements which are typically complex and sophisticated (e.g. exchanges, ATM switches, SDH Add-Drop Multiplexors). In fact, recent advances in inexpensive memory and processing capabilities suggest that the argument of simplicity is no longer valid. In summary, the Internet management framework is not well suited for telecommunications network management. Network elements with Internet management interfaces can be adapted to OSI/TMN by using the IIMC and to CORBA/TINA by using the JIDM solutions.

The OSI management framework was conceived mainly for WAN management and telecommunications environments. It is a communication framework based on the manager-agent model but has opted for sophisticated facilities in agents, as necessitated by the needs of such environments. It has adopted connection-oriented reliable transport and a fully object-oriented information model. OSI agents offer optimized multiple object access and sophisticated event management facilities that provide expressive power and minimize management traffic. The whole framework follows a "large common denominator" approach to management standardization, promoting a common style for management tasks through the system management functions which address reusability and genericity. Its object-oriented nature has led to object-oriented development environments that provide platform facilities similar to those of OMG CORBA. The key difference, however, is that the relevant APIs are not "standard" and this means that there is no application portability across different software platforms. The sophistication and complexity of the overall framework has delayed its adoption but early research efforts [Pav95a] and recent platform products have accelerated the development process and there are now a number of elements with OSI/TMN compliant interfaces in the market place. In summary, OSI management is ideally suited for telecommunication *network* management and it has been adopted as the base technology for the TMN.

OMG CORBA has evolved from the distributed system world and can be seen as a pragmatic solution that conforms to the spirit of the ISO/ITU-T ODP standards. It projects a single distributed object paradigm, accessed transparently through the ORB, as opposed to the object cluster approach of the manager-agent

model. In addition, it is mostly an object-oriented distributed programmatic framework that standardizes APIs to the underlying ubiquitous software infrastructure, the "distributed processing environment". Its object-model is fully object-oriented and largely compatible with that of OSI management. Underlying communications are reliable connection-oriented approaches, with connection management taken care of by the DPE. The event management model is simpler and less powerful than that of OSI management. Multiple object selection and discovery facilities may be supported through name servers and traders, albeit with increased traffic and reduced timeliness compared to OSI management. The use of CORBA for telecommunications network management is theoretically possible but has not yet been attempted in practice. Federation issues, e.g. for trading, have not yet fully worked out while the feasibility and cost of an ORB, name server and trader for every network element with potentially thousands of managed CORBA objects needs to be assessed. The real strength of CORBA is distributed system building and, as such, it makes it an ideal candidate for distributed service operation and management in the context of new advanced services. In summary, OMG CORBA is best suited for telecommunication distributed *service* operation and management and this is why it has been adopted by TINA as the basis of the TINA DPE.

OSI management and CORBA will have to coexist in the context of telecommunications network and service management in the years to come. We will finalize this chapter with two scenarios for their coexistence and integration: a pragmatic approach, which takes into account the past and present investment in this area, and a "blank paper" approach, potentially suitable for the long term.

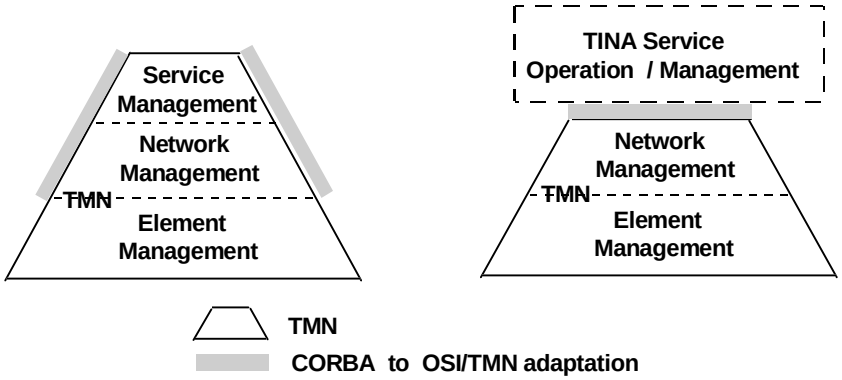


Figure 2.16 CORBA to OSI/TMN Adaptation Scenarii

TMN addresses mostly network operation and management while service management has not been addressed yet by the relevant standards groups. TMN service management addresses mostly "traditional" telecommunications services, *not* supported by distributed applications, for example leased line on demand with guaranteed quality characteristics. While TMN mechanisms cn be used to manage advanced services supported by distributed applications (e.g. video-conferencing, joint document editing), this will result in different mechanisms for service operation and service management, as already explained. The TINA framework tries to unify service operation and management mechanisms through a CORBA-based DPE and it is likely this will be the way advanced telecommunication services will be offered in the future. Despite the fact that the TINA framework intends to replace completely the TMN in the long term, the most likely scenario is that TMN will be used for *network* management and *traditional service* management while TINA will be used for *advanced service* operation and management.

The reusability of TMN management services is shown in Figure 2.16. The TMN in the left part of the figure supports both network and traditional service management. Its management services are accessible through OSI management-based X interfaces. These services could be accessed from CORBA-based environments, e.g. in customer premises networks, through suitable adapters according to the JIDM specifications. Note that adapters for other technologies, such as the World Wide Web (WWW), may be necessary. The TMN in the right part of the picture supports only network management functionality. TINA advanced services operate on top of it and reuse the supported network management services (e.g. fault/configuration management, network

quality of service management). Again, TMN services are accessed through JIDM-compliant CORBA to OSI management adapters.

The second scenario for their integration examines the possibility of combining the relative strengths of both technologies by providing OSI management facilities in a CORBA environment. An ISO/ITU-T initiative that studies the impact of ODP on OSI management is known as the Open Distributed Management Architecture (ODMA) [ODMA]. This is a theoretical high-level study. In addition, research work by the author, described in [Pav97], and by others has specified OSI-SM facilities over CORBA. The approach is depicted in Figure 2.17.

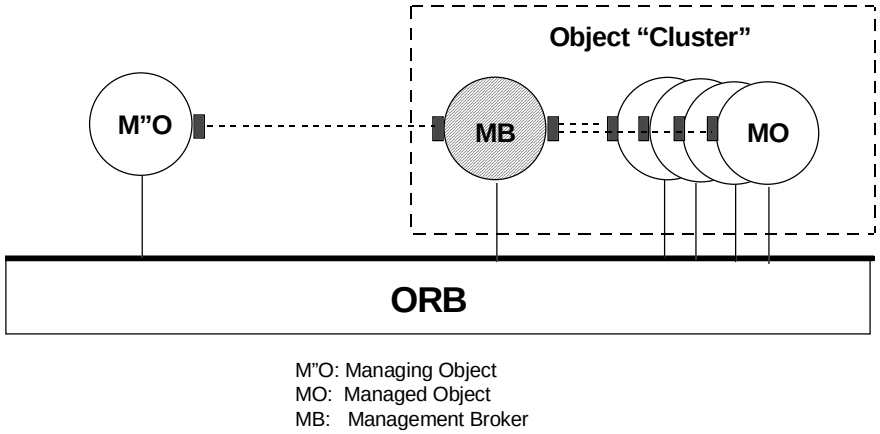


Figure 2.17 The OSI/TMN Operational Model Over CORBA

In this approach, the operational framework of OSI management is retained over CORBA through Management Brokers (MBs). Existing GDMO information viewpoint specifications are translated to IDL computational ones on a one-to-one basis, using the XoJIDM [JIDM] approach. Managed objects are implemented as equivalent CORBA interfaces with a logically bound cluster of managed objects, similar to an OSI/TMN agent, administered by a Management Broker (MB). The MB provides multiple object access facilities through scoping and filtering. In addition, it acts as an object factory, naming and notification server. Event management is provided by event forwarding discriminators and logs, with relevant filter attributes supporting the fine-grain control of notifications. The rest of the OSI SMFs are maintained as generic CORBA objects that may be instantiated within a cluster. In summary, the only necessary CORBA service is naming in order to address the MBs in a location transparent fashion.

This approach essentially maintains the OSI/TMN operational model over CORBA, but replaces the access mechanism (i.e. CMIS/P) through CORBA interactions and the distribution mechanism (i.e. OSI directory) through the CORBA naming service. Of course, interoperability protocols other than IIOP will be necessary to support interoperability in telecommunications environments; relevant mappings are expected to be produced by OMG in the future. The proposed approach retains the OSI management expressive power, event model and generic management facilities while it benefits from the distribution, portability and easy programmability of CORBA. Such an approach will make possible the eventual migration towards a single integrated "service engineering" framework that will encompass both service and network management aspects.

Acknowledgments

Sections 2.2.1 and 2.4.3 are based on earlier versions by Graham Knight of UCL, from a joint unpublished document on the comparison of the Internet and OSI System Management frameworks.

The research work for this chapter was undertaken in the context of the ACTS VITAL and REFORM projects and the RACE ICM project. The ACTS and RACE programmes are partially funded by the Commission of the European Union.

References

[X701] ITU-T Rec. X.701, *Information Technology - Open Systems Interconnection - Systems Management Overview*, 1992.

[RFC1157] J.Case, M.Fedor, M.Schoffstall, J.Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1157, 1990.

[RFC1905/6] J.Case, K.McCloghrie, M.Rose, S.Waldbusser, *Protocol Operations and Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFCs 1905 / 1906, 1996.

[X901] ITU-T Rec. X.901, *Information Technology - Open Distributed Processing - Basic Reference Model of Open Distributed Processing - Part 1: Overview*, 1993.

[CORBA] OMG, *The Common Object Request Broker Architecture and Specification (CORBA)*, Version 2.0, 1995.

[M3010] ITU-T Rec. M.3010, *Principles for a Telecommunications Management Network (TMN)*, Study Group IV,1996.

[TINA] *An Overview of the Telecommunications Information Networking Architecture (TINA)*, TINA'95 Conference, Melbourne, Australia, 1995.

[X208] ITU-T Rec. X.208, *Specification of Abstract Syntax Notation One (ASN.1)*, 1988.

[RFC1155] K.McCloghrie, M.Rose, *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC1155, 1990.

[RFC1902] J.Case, K.McCloghrie, M.Rose, S.Waldbusser, *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC1902, 1996.

[RFC1271] S.Waldbusser, *Remote Network Monitoring MIB (RMON)*, RFC1271, 1991.

[X720] ITU-T Rec. X.701, *Information Technology - Open Systems Interconnection - Structure of Management Information -Management Information Model (MIM)*, 1991.

[X721] ITU-T Rec. X.701, *Information Technology - Open Systems Interconnection - Structure of Management Information: Definition of Management Information (DMI)*, 1992.

[X722] ITU-T Rec. X.701, *Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects (GDMO)*, 1992.

[IDL] OMG, *Specification of the Interface Definition Language (IDL)*, CORBA Version 2.0, 1995.

[COSS] OMG, *Common Object Services Specification (COSS) - Event, Life-Cycle, Name, etc.*, 1994.

[X500] ITU-T Rec. X.500, *Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service*, 1988.

[X750] ITU-T Rec. X.750, *Information Technology - Open Systems Interconnection - Systems Management - Management Knowledge Management Function*, 1995.

[X710/11] ITU-T Rec. X.710/711, *Information Technology - Open Systems Interconnection - Common Management Information Service Definition and Protocol Specification (CMIS/P) Version 2*, 1991.

[Q811/12] ITU-T Rec. Q.811/812, *Specifications of Signaling System No. 7 - Q3 Interface - Lower and Upper Layer Protocol Profiles for the Q3 Interface*, 1993.

[X734/5] ITU-T Rec. X.734/735, *Information Technology - Open Systems Interconnection - Systems Management - Event Management and Log Control Functions*, 1992.

[GIOP] OMG, *General Inter-Operability Protocol*, CORBA Version 2.0, 1995.

Chapter 2

[IIOP] OMG, *Internet Inter-Operability Protocol*, CORBA Version 2.0, 1995.

[X9tr] ITU-T Draft Rec. X.9tr, *Information Technology - Open Distributed Processing - ODP Trading Function,*, 1994.

[SMF] ITU-T Rec. X.730-750, *Information Technology - Open Systems Interconnection - Systems Management Functions.*

[RFC1909/10] K.McCloghrie, G.Waters, *Administrative Infrastructure and User-based Security Model for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFCs 1909 / 1910, 1996.

[GULS] ITU-T Rec. X.830-833, *Information Technology - Open Systems Interconnection - Security, Generic Upper Layer Security (GULS), 1996.*

[OMGSEC] OMG, *Security Specification and Common Secure Interoperability - Version 1.0*, 1996.

[Bha96] S.Bhatti, K.McCarthy, G.Knight, G.Pavlou, *Secure Management Information Exchange*, Journal of Network and System Management, vol. 4, no. 3, pp. 251-257, Plenum Publishing, 1996.

[XOM/XMP] X/Open, *OSI-Abstract-Data Manipulation (XOM) and Management Protocols (XMP) Specification*, 1992.

[IIMC] NMF, *ISO/ITU-T Internet Management Coexistence (IIMC) - Translation of Internet MIBs to ISO/ITU-T GDMO MIBs and ISO/ITU-T to Internet Management Proxy*, Forum 026 and 028, 1993.

[Pav95a] G.Pavlou, G.Knight, K.McCarthy, S.Bhatti, *The OSIMIS Platform: Making OSI Management Simple*, in Integrated Network Management IV, ed. A.S.Sethi, Y.Raynaud, F.Faure-Vincent, pp. 480-493, Chapman & Hall, London, 1995.

[Pav95b] G.Pavlou, K.McCarthy, S.Bhatti, N.DeSouza, *Exploiting the Power of OSI Management in the Control of SNMP-capable Resources Using Application-level Gateways*, in Integrated Network Management IV, ed. A.S.Sethi, Y.Raynaud, F.Faure-Vincent, pp. 440-453, Chapman & Hall, London, 1995.

[JIDM] X/Open / NMF, *Joint Inter-Domain Management (JIDM) Specifications - SNMP SMI to CORBA IDL, ASN.1/GDMO to CORBA IDL and IDL to GDMO/ASN.1 translations*, 1994.

[ODMA] ITU-T Draft Rec. X.703, *Open Distributed Management Architecture*, 1995.

[Pav97] G.Pavlou, *From Protocol-based to Distributed Object-based Management Architectures*, Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Sydney, Australia, 1997.