From Protocol-based to Distributed Object-based Management Architectures

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

Abstract: OSI Systems Management (OSI-SM) and Internet SNMP have been conceived as protocol-based management frameworks, adopting the manager-agent model that governs interactions between management applications. The manager-agent model is object-oriented in information specification terms but addresses only "on the wire" interactions, leaving deliberately unspecified aspects relevant to the internal structure of management applications. On the other hand, OMG CORBA and similar technologies can be thought as pragmatic counterparts of Open Distributed Processing (ODP). They project an object-oriented paradigm and address mainly programmatic interface aspects, while they use a general purpose Remote Procedure Call (RPC) protocol as opposed to a specialized management protocol.

The benefits of using distributed object technologies for management are easy programmability, multiple programming language bindings, application portability and distribution. On the contrary, management protocol approaches offer optimized mechanisms for accessing management information, including event dissemination, and do not constrain implementations. This paper examines the basic characteristics of the two types of architectures, explains their relevant similarities and differences and proposes a future architecture based on distributed objects. This retains the operational paradigm of OSI-SM model over the relevant distributed object framework, bridging the two worlds and maintaining the best aspects of both.

Keywords: Manager-Agent, Distributed Objects, OSI-SM, ODP, OMG CORBA

1. Introduction and Overview

OSI Systems Management (OSI-SM) [X701] and Internet SNMP [SNMP] have adopted the manager-agent paradigm. Physical or logical real resources at different levels of abstraction are modeled as managed objects, encapsulating the underlying resource and offering an abstract access interface at the object boundary. The management aspects of entities such as network elements and distributed applications are modeled through “clusters” of managed object instances, seen collectively across a management *interface*. The latter is defined through the formal specification of the relevant managed object types or classes and the associated access mechanism i.e. the management access service and supporting protocol stack. Manager applications access managed objects across interfaces in order to implement management policies. Distribution and discovery aspects are orthogonal to management interactions and need to be supported by other means. Both OSI and Internet SNMP 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 depicted in Figure 1. Manager and agent applications contain other internal objects which support the implementation of relevant functionality. These are

not visible externally, so they are depicted with dotted lines. The term “agent” refers essentially to the object cluster made visible across a management interface, so the agent does not explicitly manifest itself. 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 those models, managed objects exist also 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 and service management according to the TMN model. Management applications may act in dual manager-agent roles, in either peer-to-peer or hierarchical relationships.

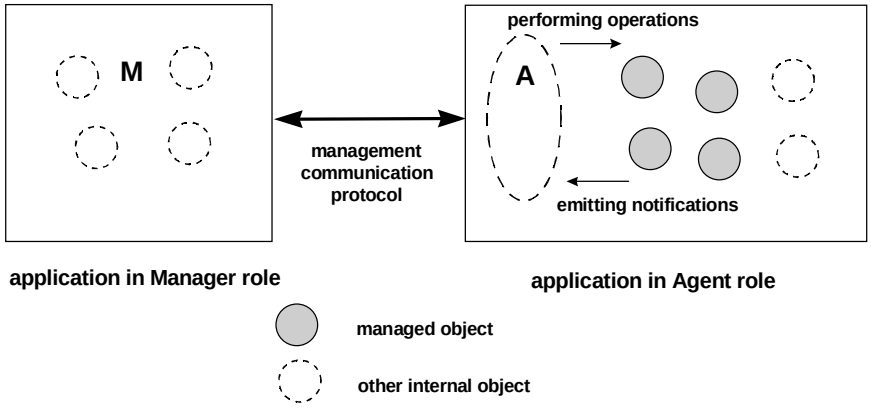


Figure 1 The Manager-Agent Model

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 i.e. the Object Request Broker (ORB). Server objects are accessed through interfaces on which operations are invoked by client objects in a location transparent fashion. The choices made by 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” vs. “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.

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. 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 it could be also used for management.

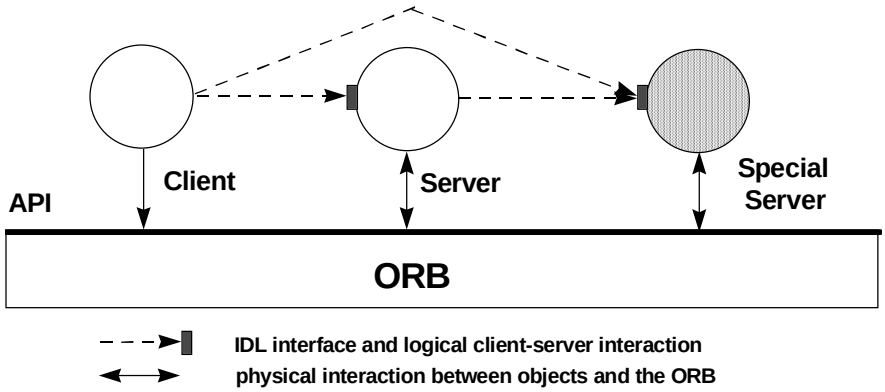


Figure 2 The OMG CORBA Model

In this paper we compare the OSI Systems Management and ODP / OMG CORBA technologies in the context of telecommunications network and service management and propose a new architecture that retains the best aspects of both. In section 2 we examine information modeling aspects while in section 3 we examine the access and distribution aspects, including the suitability of CORBA for network management. In section 4 we present the proposed architecture and finally we present a summary and conclusions.

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 be also manageable. The ideal information model must cope easily with management information related to such a variety of management targets. The original motivation for the development of the two frameworks was different: network / service management for OSI-SM, distributed application operation and management for OMG CORBA.

2.1 The OSI Management Information Model

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 which defines the possible containment relationships between instances of those classes. There may be many types of relationships between classes and their instances but containment is treated as a primary relationship and is used to yield unique names. The smallest re-usable entity of management specification is not the object class, as is the case in other O-O frameworks, but the *package* which is a collection of attributes, actions, notifications and behavior. Object classes are characterized by one or more mandatory packages while they may also comprise conditional ones.

Actions accept arguments and return results, providing a 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. Object classes may also have associated notifications, specifying the condition under which they are emitted and their syntax. 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. OSI SM follows a fully O-O paradigm and makes use of concepts such as inheritance and polymorphism. The use of class inheritance allows re-usability and extensibility of both specification and associated implementation.

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 specifies formally only syntactic object-oriented aspects of managed object classes. Semantic aspects, i.e. the contents of behavior templates, are expressed currently in natural language.

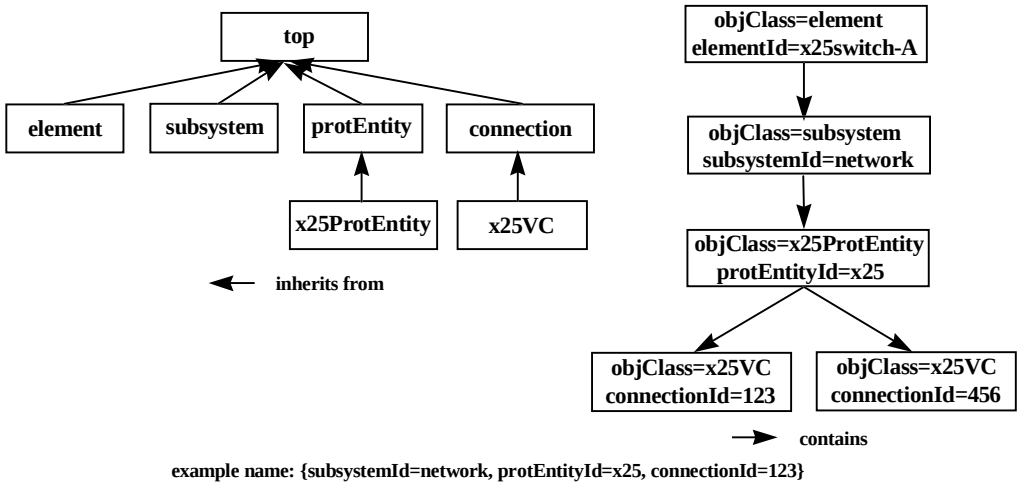


Figure 3 Example OSI Inheritance and Containment Hierarchies

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) e.g. connectionId=1234. 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. An example of a containment tree and an object name is shown in Figure 3. OSI management names are assigned to objects at creation time and last for the lifetime of the object.

2.2 ODP / OMG CORBA Information Model

The ODP / OMG CORBA information model is fully object-oriented to 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 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. OMG interfaces are specified using the Interface Definition Language (IDL) [IDL]. IDL may be thought as broadly equivalent to the GDMO/ASN.1 combination in OSI management, though less powerful.

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*. Attributes accept Get and possibly Set operations while only standard exceptions may signify an error during such operations. This is in contrast to GDMO, where arbitrary class-specific errors 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.

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. In fact, 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 difference is that in IDL there is no formal way 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. There are more differences with respect to the way events are disseminated, discussed in section 3.

ODP / OMG do not provide a built-in operation for instantiation of interfaces by client or managing objects. 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 also flexibility in the placement of new objects.

While the OMG IDL object model has many similarities to GDMO, a marked difference concerns naming. OMG objects can be identified and accessed through *Object References*. The latter are assigned to objects at creation time and are opaque types i.e. have no internal structure and, as such, do not reveal any information about the object. An object may have more than one reference. OMG objects may also be assigned names. The latter are distinct from objects, unlike OSI-SM where an object has always 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. The name server may be essentially used to assign names to objects and to resolve names to object references.

CORBA objects may be used to model in a one to one mapping 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 name server or the trader, as discussed next.

3. Access and Distribution Paradigm

In the two 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-SM and OMG CORBA there are marked differences in the relevant access paradigms. OSI-SM follows 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. On the other hand, OMG CORBA specifies the API to the Object Request Broker 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 brokering, name serving and trading.

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]. On the other hand, OMG CORBA is influenced by ODP [X901] and, as such, it has been designed from the beginning 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-SM has targeted the optimal support for managed object access services.

3.1 OSI Management

OSI Management uses a connection-oriented reliable transport. The OSI management service/protocol (CMIS/P) [X710/11] operate over a full 7 layer OSI stack and rely on the reliable OSI transport service, which can be provided over a variety of transport and network protocol combinations. 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 below that object through scoping. Objects of individual levels, until individual levels or the whole subtree may be selected. The selection may be further eliminated through a filtering parameter that specifies a predicate based on assertions on attribute values. 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. When applying an operation to multiple objects through scoping and filtering, 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. Figure 4 depicts the interactions between applications in manager and agent roles using CMIS.

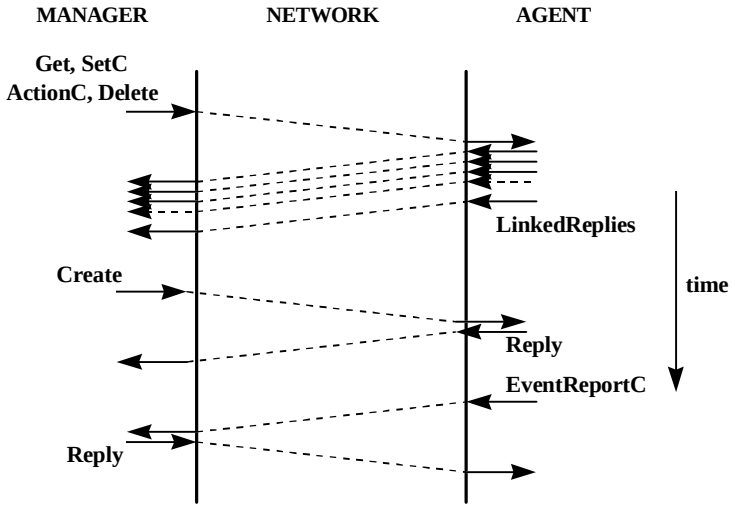


Figure 4 CMIS Interactions

The event reporting model in OSI management is very sophisticated, allowing for very fine control of emitted notifications. Special support objects known as Event Forwarding Discriminators (EFDs) [X734/5] 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. In addition, an emitted notification may be logged locally by being converted to a specific log record contained in a log object previously created by a manager. Log control is exercised again through a filter attribute specified in the log object [X734/5].

Distribution aspects in OSI management are supported by the OSI directory [X500]. OSI management applications are represented by directory objects which contain addressing and capabilities information [X750]. Managed objects in agents may be addressed by global names starting from the top of the directory information tree. This naming architecture assumes a “logical” link between the topmost MIT object of an agent and the corresponding directory object. Through this approach, the OSI directory and management name spaces are essentially unified. Global names guarantee location transparency as they remain the same even if the management application changes location.

3.2 ODP/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 specified also a Remote Procedure Call (RPC) protocol over the Internet transport stack; this known as the Internet Inter-ORB Protocol (IIOP) [IIOP]. In fact, the ODP *access* transparency prescribes independence of the underlying access protocol and CORBA provides both independence and portability due the agreed APIs. The access protocol could change without any effect on application software.

The agreed CORBA protocol is a connectionless RPC one that uses TCP and IP as transport and network protocols respectively. This means that it benefits from the reliable transport TCP provides but it does not require the establishment of connections by applications, according to the connectionless nature of RPC protocols. Connection management is dealt with within the RPC layer and some caching is necessary for optimization purposes. The CORBA RPC protocol is a *request/response* type of protocol in which the exact structure of the request and response packets are defined by the IDL specification of the accessed CORBA interface. There are no special facilities built in the protocol for object discovery/naming and multiple object access in a similar fashion to the OSI management naming, scoping and filtering. Such facilities are provided instead in a limited fashion by the ORB and by special objects known as *servers*. In summary, the CORBA RPC protocol provides a *single* object access mechanism with higher-level facilities are provided by standard OMG servers [COSS].

The CORBA operational paradigm is different to 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 e.g. printer servers, statistical calculation servers, etc. 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 though that the whole framework is optimized towards a "*single 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 location-transparent manner. The ORB will find an instance of that type and return an object reference to the client object. Instances of the same type can be distinguished though 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. The trader [X9xx] 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 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 found 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 federated in order to be able 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.

3.3 The Use of CORBA for Network / Service Management

Let's examine now how CORBA could be used for network and service management, contrasting its approach to the protocol-based OSI-SM approach. If CORBA is used as the underlying access and distribution mechanism, managed objects could 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 provided 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.

The problem with the use of CORBA 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, e.g. 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 generated management traffic will be at least twice that of OSI management. With 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.

In the current state of CORBA, with federation issues far from being worked out, generated traffic will be much bigger since managed objects will have to contact servers across the network. We will use an example to demonstrate the amount of traffic generated by both OSI SM and CORBA for network management. Let's assume a network element such as an ATM switch which contains N managed objects representing established Virtual Channel Connections (VCCs). Out of those, M managed objects originate from a particular source

address (N > M). If an OSI-SM manager wants to find those VCCs that originate from that address, it will have to locate the element agent through the directory and send a request with scope and filter constraints. The overall number of application level packets, ignoring connection establishment, will be: 2 for contacting the directory and 2+M for retrieving the objects [M+4 in total]. The same amount of traffic with CORBA will be: 2*N for the VCC objects to contact the trader, 2 for the client to contact the trader and 2*M for the client to retrieve the results [2*(N+M+1) in total]. In addition, it the specified filter asserts on a dynamically changing attribute instead of a static property, the trader will have to generate another 2*N packets searching through the VCCs.

4. A Unified CORBA-based Architecture

As explained in the previous section, the use of the ODP / OMG CORBA model for network and service management presents a number of difficulties to overcome due to the dynamic nature of management information and the number of managed objects typically present in managed elements. On the other hand, the OSI-SM model scales much better and has already been used for managing large telecommunications infrastructures e.g. SDH/SONET, ATM, etc. Based on this observation, an ideal framework would combine the expressive power of OSI-SM and the programmability, portability and distribution aspects of ODP / OMG CORBA. In order to specify such a framework it is important to be able to map management information specifications in GDMO to computational interfaces in IDL. As described in section 2, the information modeling aspects of the two frameworks are similar and the X/Open Joint Inter-Domain Management task force (XoJIDM) has defined rules for this mapping [JIDM], which we describe next.

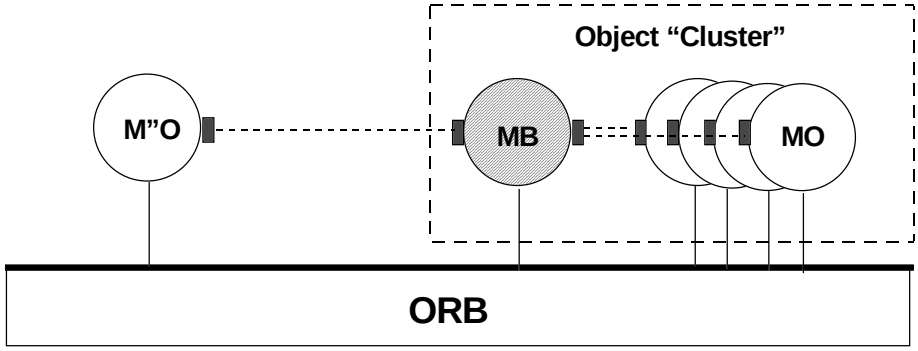
Mapping GDMO specifications to CORBA IDL is not straightforward because GDMO as information specification language and CMIS/P as the access method have a number of aspects for which there exist no IDL and CORBA equivalents. These include the late binding of functionality to managed object instances through the use of conditional packages; the fine grain support for event discrimination; and the use of scoping and filtering as “query language” facilities that may result in multiple replies.

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, etc. GDMO actions may be naturally mapped onto IDL methods. 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 though becomes an implementation issue: the standard CORBA *not_implemented* exception should be raised whenever a method of a non-implemented package is invoked.

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 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 ITU-T GDMO information modeling working groups to avoid the use of conditional packages in a non IDL-compatible fashion.

A more important difference concerns 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 in clusters can be accessed collectively through the CMIS/P scoping and filtering facilities. These may be used for discovery services e.g. "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 and this is not only an engineering-level optimization but allows as well a higher level of abstraction to be provided to managing functions. Discovery facilities may be provided through naming servers and traders in CORBA as discussed in section 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 over CORBA.



M"O: Managing Object
MO: Managed Object
MB: Management Broker

Figure 5 The OSI-SM Operational Model Over CORBA

This is exactly the approach we have followed. In this, the operational framework of OSI management is retained over CORBA through Management Brokers (MBs). A logically bound cluster of managed objects similar to an OSI/TMN agent application is administered by a management broker; the latter provides multiple object access facilities similar to CMIS. Of course, managed objects may be also accessed directly in the standard CORBA fashion. Event management is provided by event discriminators and logs through filtering, in order to overcome the relevant CORBA limitations. Finally, the rest of the OSI Systems Management Functions [SMF] are maintained as generic CORBA objects that may be instantiated within a cluster. This approach essentially maintains the OSI operational model over CORBA but replaces the access (i.e. CMIS/P) and distribution (OSI directory) mechanisms. As such, it retains the OSI management expressive power, event model and generic management facilities while it benefits from the distribution, portability and easy programmability of CORBA. The approach is depicted in Figure 5.

We have actually specified and implemented Management Brokers as described above. A Management Broker act as object factory, name server, trader and notification server using the relevant OSI-SM methods and techniques i.e. CMIS-like object creation, naming

through distinguished names, scoping / filtering and event reporting / logging through event discriminator and log objects. As an example, the following is a fragment of the IDL specification of one of the Management Broker interfaces. The example demonstrates two operations: the first, objectSelection, allows a client to identify information objects according to certain scope and filter parameters; the second, multipleObjectGet, allows a client to read selected attributes of a group of objects in a single operation.

```
// hierarchical naming in X.700 style

typedef struct RelativeName_t {
      AttributeId_t attrId;
      string         attrValue;
};
typedef sequence<RelativeName_t> DistinguishedName_t;
typedef DistinguishedName_t ObjectInstance_t;

// object selection through scoping and filtering

enum ScopeChoice {
      baseObjectChoice,
      firstLevelOnlyChoice,
      wholeSubtreeChoice,
      individualLevelChoice,
      baseToNthLevelChoice
};
typedef struct Scope_t {
      ScopeChoice choice;
      unsigned long level;
};

// Filter_t is a translation of the X.711 CMISFilter in IDL

typedef struct ObjectSelection_t {
    Scope_t scope;
    Filter_t filter;
};

interface MultipleOpManagementBroker : ManagementBroker {

                void objectSelection (
                          in ObjectInstance_t baseObjectInstance,
                          in ObjectSelection_t objectSelection,
                          out ObjectInstanceList_t objectInstanceList
                ) raises (OBJECT_SELECTION_ERRORS);

                void multipleObjectGet (
                          in ObjectInstance_t baseObjectInstance,
                          in ObjectSelection_t objectSelection,
                          in AttributeIdList_t attributeIdList,
                          out GetResultList_t resultList
                ) raises (MULTIPLE_OBJECT_OP_ERRORS);

                // ...
}
```

A Management Broker does not only have a CMIS-like interface but also interfaces to be “told” about new objects it needs to administer and to receive notifications from the managed objects it administers. In addition, every MO inherits from a *ManagedObject* interface through which the MB notifies it that it administers it. A MB and the objects it administers may be physically distributed but they logically form a “cluster” which can be collectively accessed. A managed object may belong to more than one MB domains. In the case of managed network elements, at least one MB needs to be physically located together with the local managed objects so that it provides optimized access and event dissemination facilities with minimal management traffic. In general, it is not necessary to provide separate IDL interfaces for every managed object as this may not be technologically feasible with the current state of CORBA implementations. In this case, the managed objects and the broker

may interact through a local mechanism e.g. internal C++/Smalltalk interfaces if they share a common address space (an engineering “capsule” in ODP terms).

We have actually implemented a first version of such a management broker using the OSIMIS [Pav95] platform and the Orbix implementation of CORBA. We have used this to provide clusters of objects that represent Network Topology Configuration Maps (NTCMs) which model ATM Virtual Path and Virtual Channel layer networks. In those applications, objects are “lightweight” representations of element level managed objects and represent network-wide topological information through containment and other relationships. Those “maps” can be flexibly navigated through scoping and filtering while topology changes may be effected through object creation and deletion. We have actually trialled those applications in the context of a real TINA system [TINA]. The prototype implementation served to validate and demonstrate the architectural concepts presented.

5. Summary and Conclusions

In this paper we have presented a new management architecture which retains the operational model of OSI-SM/TMN over a CORBA-based Distributed Processing Environment. We have actually specified and implemented Management Brokers that mirror the facilities of OSI-SM but use CORBA as the access and distribution mechanism. Our approach is based on the results of the XoJIDM work for mapping GDMO specifications to IDL but it proposes a native CORBA-based Open Distributed Management Architecture (ODMA). In fact, there exists an ISO/ITU-T initiative that studies the impact of ODP on OSI management, also known as the Open Distributed Management Architecture (ODMA) [ODMA]. Our concrete specification will be passed as input to that group.

While there is plenty of ongoing research regarding OSI-SM/TMN and CORBA migration and interworking, we believe our approach retains the relevant advantages of TMN for network management while it is both compliant and complementary to the JIDM approach. In addition, this is a viable path for gradually migrating existing TMN systems over CORBA-based DPEs. The approach will be also proposed to bodies such as TINA, JIDM, OMG and the ITU-T Study Group IV. An additional aspect of the presented approach is that it can also be used to provide generic adaptation functions to existing OSI-SM/TMN systems. In fact, we have implemented both native CORBA-based MBs but also generic MB adapters to OSI-SM/TMN systems. As such, our approach provides a smooth migration path from current TMN-based management systems to target systems operating over CORBA-based DPEs. This will make possible a single integrated “service engineering” framework that will encompass both network management and service operation / management aspects.

Acknowledgments

This paper describes work undertaken in the context of the ACTS REFORM (REsource Failure and RestORation Management) and VITAL (Validation of Integrated Telecommunication Architecture for the Long term) projects which are partially funded by the Commission of the European Union. The author would also like to acknowledge Thurain Tin of UCL who implemented the generic adapter version of the Management Broker and the XoJIDM team for their excellent work.

References

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

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

[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)*, 1991.

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

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

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

[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)*.

[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.

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

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

[X9xx] ITU-T Rec. X.9xx, *Information Technology - Open Distributed Processing - Trader*.

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

[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.

[Pav95] 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.

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