4. Mapping the OSI-SM / TMN Model Onto Emerging Distributed Object Frameworks

4.1 Introduction

Chapter 4 of this thesis examines the possibility of using emerging distributed object technologies as the basis for the TMN. Since the early inception of Open Distributed Processing, a number of related technologies tried to provide a uniform and ubiquitous framework for building distributed applications. The latest and most powerful of those technologies is OMG CORBA, which can be thought as the pragmatic counterpart of ODP. Since the TMN is a large scale distributed system, it is valid to consider its mapping onto CORBA, considering the latter as the representative distributed object technology. This implies replacing OSI-SM and the OSI Directory with OMG CORBA as the base technology for the TMN. The relationship between OSI-SM and OMG CORBA has attracted considerable attention from the research community in recent years. In this chapter we propose a solution that maintains the expressive power of OSI-SM and provides a smooth migration path towards a CORBA-based TMN.

A key motivation for using CORBA in TMN environments is the following. OSI-SM was conceived as an object-oriented management technology in the absence of a general purpose distributed object-oriented framework. OMG CORBA provides exactly such a framework and it is likely that it will be used in the future for supporting advanced telecommunications services e.g. those conforming to the emerging Telecommunications Networking Information Architecture (TINA) [TINA]. Using OSI-SM to manage distributed components of those services will result in a discrepancy of technologies: one technology for the service operation and control, i.e. OMG CORBA, and another technology used for managing the relevant service components, i.e. OSI-SM. OMG CORBA may provide the unifying framework, resulting in economies of scale.

Additional motivations for using OMG CORBA in TMN systems are the following. CORBA provides a superior distribution paradigm, in which every object could be potentially distributed. In OSI-SM only whole agent applications can be distributed. CORBA performance could be

potentially better than OSI-SM due to a more lightweight protocol stack. Finally, CORBA exhibits multiple O-O programming language mappings while both OSIMIS [Pav93b] and the NMF TMN/C++ [Chat97] support mainly C++ APIs. On the other hand, OSI-SM exhibits a more expressive object model and superior object access and event dissemination mechanisms, so the mapping between the two presents a difficult technical challenge.

This chapter is organised as a “super-chapter”, in a similar fashion to chapters 2 and 3 of this thesis. Related research work is presented throughput the various sections, in a similar fashion to previous chapters.

Section 4.2 describes Open Distributed Processing, which underpins distributed object technologies. This introduction serves as state of art but also goes further, positioning OSI-SM and TMN in the ODP context and examining them from the ODP viewpoints.

Section 4.3 describes different incarnations of ODP-based technologies, namely ANSA, the OSF DCE and OMG CORBA and considers their suitability as TMN technologies. This is again more than a state of the art description, presenting those technologies as candidates for the TMN and examining them against the properties of distributed frameworks identified in Chapter 2.

Section 4.4 considers in detail how CORBA can be used as the base technology for the TMN. A pure ODP-oriented approach is presented first, which is practically unfeasible with the current state of CORBA technology. Then a more pragmatic mapping approach follows, starting from a minimal solution and adding gradually features so that the full power and expressiveness of OSI-SM is recaptured. This mapping represents the key research contribution of this chapter and has been partly validated through implementation.

Section 4.5 presents a performance analysis and evaluation of the CORBA-based framework, trying to identify the performance characteristics of CORBA and potential advantages over OSI-SM. This section though does not go though into the same level of detail as the relevant OSI-SM analysis in Chapter 3.

Finally, section 4.6 presents a summary and highlights the research contributions of this chapter.

4.2 Open Distributed Processing

The ISO/ITU-T Open Systems Interconnection (OSI) and the IETF standardisation efforts addressed the problem of heterogeneous system interconnection through standardised protocols. The application layer in both frameworks addressed partly the needs of distributed application development through the OSI ROSE [X219] and ASN.1 [X209] and the Internet Sun RPC [Sun88] and XDR [Sun87]. Both ROSE and RPC systems are object-based as opposed to object-oriented. In addition, ROSE is not a software framework and as such it does not support application software portability through standard APIs.

Towards the mid to late eighties it became clear that the development and deployment of distributed applications could be facilitated by software infrastructures with standard APIs, in addition to the required standard protocols for interoperability. The Advanced Networked Systems Architecture (ANSA) [ANSA89a][ANSA89b] was a research and development effort towards a distributed software platform which aimed to provide well-defined APIs for application components, facilitating their rapid development and achieving application portability. In fact, ANSA was more than a distributed software environment: it introduced a set of concepts for distributed systems and also influenced the ISO/ITU-T activity on the standardisation of Open Distributed Processing (ODP) [ODP].

It should be noted that the ISO/ITU-T OSI Systems Management framework was the only OSI application layer effort that adopted a fully object-oriented information specification approach. The realisation framework presented in Chapter 3 proposes a software infrastructure that addresses the issues of rapid application development and software portability through well-defined APIs. This environment has a lot in common with distributed software frameworks based on ODP principles, as it was explained in Chapter 3. Both ANSA and the embryonic ODP framework of the late eighties were important influences for the author. On the other hand, the proposed software framework is specific to object-oriented OSI-SM/TMN systems while ANSA / ODP are more general frameworks, targeting general purpose distributed applications.

4.2.1 The ODP Model

ODP aims to provide both an architectural framework for distributed systems and also associated implementation technology for distributed objects. A standardised software infrastructure will facilitate the integrated support of distribution and will achieve portability and interoperability. The essence of the ODP framework is demonstrated in Figure 4-1. The ODP software platform

operates over the native computing and communications environment in the various network nodes, hiding heterogeneity and providing an homogeneous view of the underlying resources. Objects "sitting" on this distributed platform interact with each other while they are shielded from the problems and complexity of distribution through various transparencies.

ODP targets a programmatic interface between objects in *client* and *server* roles and the underlying support environment i.e. the ODP platform. The latter is shown as crossing the boundaries of network nodes in Figure 4-1 because of the fact it hides heterogeneity and provides a uniform view of the underlying resources. Parts of the global ODP platform may be provided by different vendors while objects will get a uniform view through the standardised APIs. Another commonly used term for such a software platform is *middleware*, since it is an enabling technology.

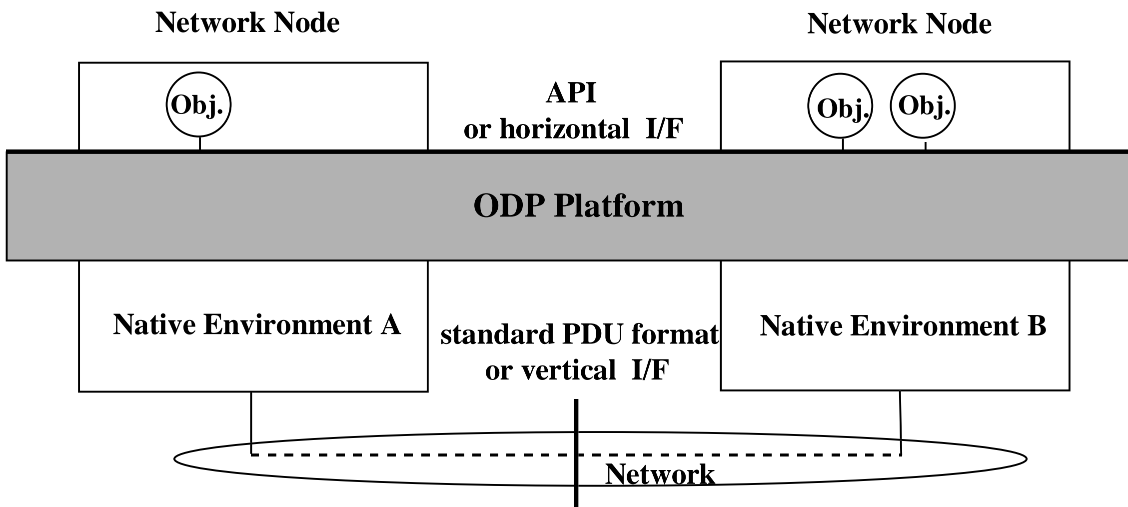


Figure 4-1 The ODP Framework

The approach of standardising APIs for software portability and uniformity is sometimes referred to as "horizontal" standardisation. OSI and the Internet address interoperability through standard protocols which result in agreed message formats; this approach is sometimes referred to as "vertical" or "on the wire" standardisation. ODP systems need to agree on standard protocols in order to interoperate. It should be possible though to port an ODP platform over different underlying communications protocols without any changes to the APIs.

The ISO/ITU-T ODP Reference Model (ODP-RM) [ODP] is split into four parts, all of which use object-oriented concepts. Part 1 [X901] provides the overview, introduces the concept of information distribution, discusses the issues it raises and proposes a number of distribution transparencies to cope with those. Part 2 [X902] is descriptive and introduces modelling,

specification and architectural concepts. Part 3 [X903] is prescriptive and provides the concepts, rules and functions a system must deal with in order to be ODP conformant, which are expressed in terms of five *viewpoints*. Finally, Part 4 [X904] attempts to map the modelling concepts of part 2 onto formal description techniques applicable to the five viewpoints.

4.2.2 The ODP Distribution Transparencies

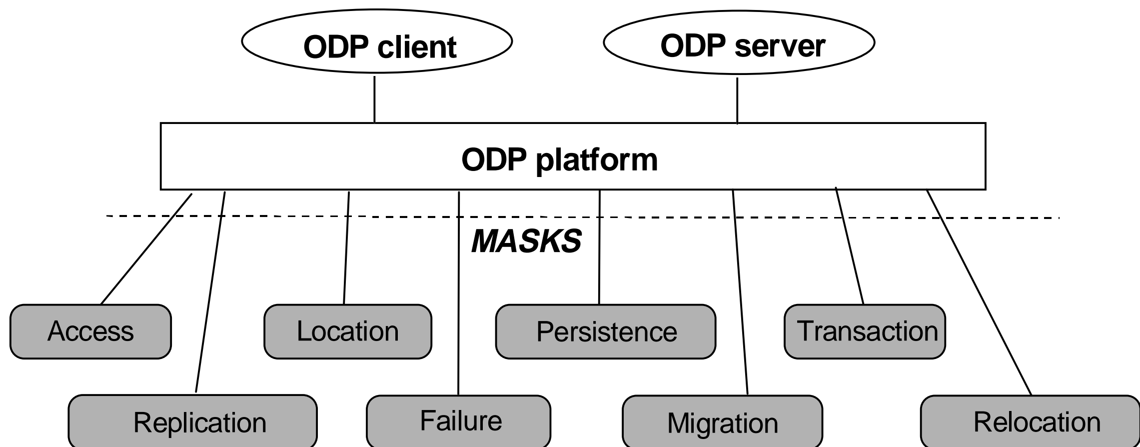


Figure 4-2 ODP Distribution Transparencies

An ODP platform supports the interaction between objects in client and server roles, through *interfaces* supported by the server object i.e. this is an asymmetric relationship in the same fashion to the manager-agent relationship explained in Chapter 2. The ODP platform provides the following transparencies to client objects, shielding them from the problems of distribution:

- *access* transparency, which masks heterogeneity in computer architectures, programming languages, data representations and invocation mechanisms;
- *location* transparency, which masks the topological details when “binding” to distributed server objects;
- *persistence* transparency, which masks the activation and deactivation of a server object with respect to its persistent secondary storage representation;
- *transaction* transparency, which masks the coordination of activities across a number of server objects in order to achieve overall consistency of operations;
- *replication* transparency, which masks the existence of a number of server objects with the same properties used for performance, availability, reliability and fault tolerance;

- *failure* transparency, which masks the failure of objects and their subsequent reactivation or substitution through replication, migration etc.;
- *migration* transparency, which masks the fact that a server object may change location because of performance, security or other reasons; and
- *relocation* transparency, which is related to migration and masks the relocation of a server object while clients are still bound to it and operations may be in progress.

Access, location, persistence and transaction transparencies are not only pertinent to ODP systems. The OSI-SM/TMN object-oriented distributed platform presented in Chapter 3 exhibits access, location and persistence transparencies while transaction transparency could be added through the ISO/ITU-T Distributed Transaction Processing (DTP) [DTP] framework. Failure, replication, migration and relocation transparencies are much more difficult to provide. In fact, such transparencies are not provided by any existing ODP-influenced system, e.g. OMG CORBA [CORBA] or Microsoft DCOM [DCOM].

4.2.3 The ODP Trading Function

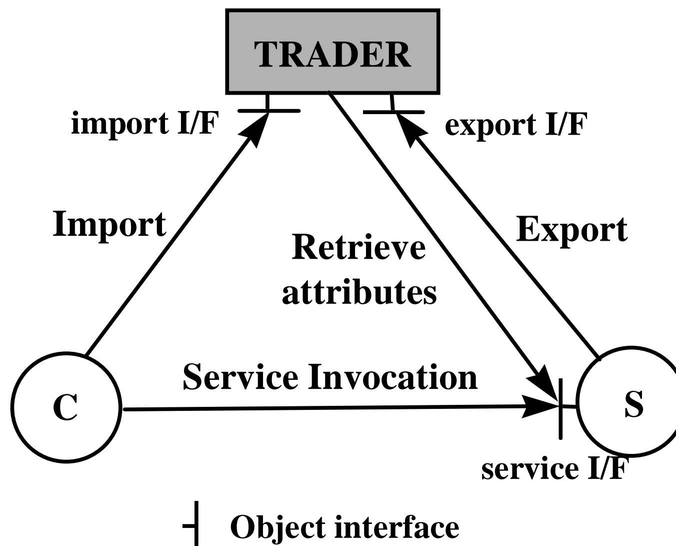


Figure 4-3 The ODP Trading Function

Objects "floating" on the ODP platform need to be able to locate each other before invoking distributed operations, which are supported by a server object's *interfaces*. Discovery of a server object is supported by the ODP trading function which is depicted in Figure 4-3. The ODP *Trader* [ODP][X9tr] is a special server that holds information about a "service space", manifested by the interfaces of various types of server objects. The latter *export* information to

the trader about the interfaces they support and associated static *properties* or dynamic *attributes*. Properties are fixed for the lifetime of an object instance, e.g. the "next hop address" of a route object, while attributes may change dynamically, e.g. the current number and size of jobs in a printer queue.

Client objects wishing to contact a particular type of server interface contact the trader and pass a request that may include assertions on interface properties and/or attributes. The trader may need to retrieve the asserted attributes from all the objects with interfaces of the required type, evaluate the client's request and pass back the matching interface *references*. The function of the trader is in fact similar to the OSI-SM filtering function, but only test for equality is possible: this is because ODP interface attributes do not have "MATCHES FOR" properties in a similar fashion to GDMO attributes. Also from an engineering perspective, filtering is tightly coupled with the OSI-SM MOs while the ODP trader is a separate server object that may operate on a different node. Traders need to be federated in order to be able to cope with big service spaces and different administrative domains. Issues of trader federation have not yet been addressed in detail. The use of the OSI Directory [X500] as supporting technology is mentioned in the relevant recommendations since it supports federation.

Trading is generally important in distributed systems for locating server objects that satisfy particular criteria, e.g. locate the printer server with the smallest job queue. On the other hand, a subset of trading is mostly used in distributed environments, known as *name serving*. Manager or client objects need to be able to locate managed or server objects by name, given the fact that there will typically exist more than one instance of a particular type of managed object in the distributed environment. Though ODP standards do not explicitly mention name serving, associated technologies such as the OSF Distributed Computing Environment (DCE) [DCE] and OMG CORBA [CORBA] provide name serving facilities. A Name Server (NS) is a simplified form of a trader which only stores the name properties of server interfaces and resolves supplied names to interface references. The reader is reminded here that name serving in OSI-SM / TMN environments is supported by the OSI Directory, as described in section 2.4 of Chapter 2. In this case the server object is the application in agent role which exports its "global" name to the directory.

The trader depicted in Figure 4-3 exhibits two distinct interfaces: an export interface used by exporters or servers and an import interface used by importers or clients. In general, ODP objects may have more than one interfaces, bound to a common state through the object. In fact, ODP objects are characterised by the interfaces they support, which means that object interfaces, as

opposed to objects themselves, are the "first class citizens" of the ODP "society". It should be noted that the only ODP-based technology that supports multiple distinct interfaces per object is the MS DCOM.

4.2.4 The ODP Viewpoints

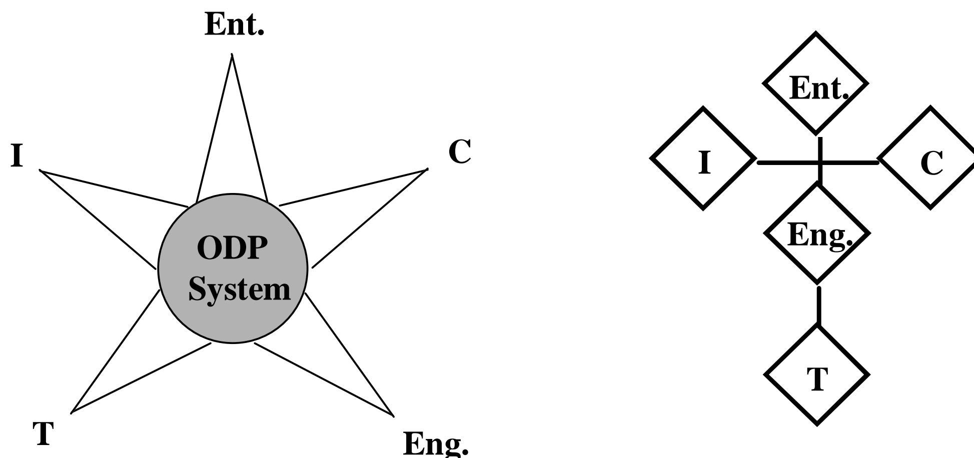


Figure 4-4 The ODP Viewpoints

Another important aspect of ODP is its specification methodology. The ODP prescriptive model [X903] proposes the use of five *viewpoints* which, when considered together, provide a complete framework for distributed system specification. A viewpoint is a form of abstraction of the overall system, focusing on particular concerns. The five ODP viewpoints are the following:

- the *Enterprise* viewpoint, which focuses on the business view of the system, including the purpose, scope, services offered, actors involved, their roles and business policies;
- the *Information* viewpoint, which focuses on the information content of the system, including the semantics and relationships of information objects; the information properties of a system constitute its information model;
- the *Computational* viewpoint, which comprises a description of the system in terms of interacting objects, focusing on the functional decomposition of the system for the purpose of distribution;
- the *Engineering* viewpoint, which focuses on the functions required to support distribution, including the provision of distribution transparencies by the ODP platform; and

- the *Technology* viewpoint, which focuses on real world artefacts from which the system is constructed i.e. hardware, software, operating systems, databases, etc.

Figure 4-4 depicts two views of the five viewpoints. The left view emphasises the fact that each viewpoint projects a different view of the same system. The right view shows a hierarchical relationship of the viewpoints: the enterprise view comes first, representing the "raison d' être" of the system; the system design follows through the information and computational decomposition; the system is subsequently deployed in the engineering viewpoint; finally, the technology viewpoint documents the technology aspects.

Each viewpoint may have one or more associated viewpoint languages that define the concepts and rules for representing the system from that viewpoint. Natural or formal, textual or graphical notations may be used as required by that particular viewpoint. An important issue is that the overall specification of the system from the five viewpoints should be consistent.

The aspects of the enterprise viewpoint are typically documented in natural language. In the case of the TMN, the description of the management services [M3200] and the relationships between the administration operating the TMN and other administrations or human users can be thought as part of the enterprise viewpoint.

The information and computational viewpoints are intimately coupled, since they both express the design of the system. On the other hand, it is the computational viewpoint that defines the distributed interactions between objects and the structure of messages exchanged. In the simplest case, there is a one-to-one mapping between information and computational objects. In other cases, a computational object may hold information objects internally, providing access to them through its computational interface. The relationship between the ODP information and computational viewpoints is generally a very complex issue and not particularly well addressed. The TINA information modelling concept document [Crist95] includes an interesting discussion on this relationship, concluding that no general rules can be drawn. We will investigate in detail this separation in the context of OSI-SM / TMN later in this chapter.

The OSI-SM GDMO [X722] and GRM [X725], despite the fact they were developed for the purpose of defining managed objects and their relationships, represent formal techniques for general information modelling. A slightly modified version of those known as Quasi-GDMO/GRM is used as an information modelling language in TINA [Crist95]. Another widely-used notational technique for describing information models in the ODP information viewpoint is the Object Modelling Technique (OMT) [Rumb91]. This has also been used throughout this thesis for describing static class relationships (see Appendix B).

Object interfaces are specified in the computational viewpoint using an Interface Definition Language (IDL). The methods of the relevant interfaces and their arguments define the messages that can be sent to the objects that support those interfaces. Two types of interfaces can be supported by ODP objects: *operational* interfaces which support request-response operations; and *stream* interfaces, which support continuous information flows, e.g. for audio and video streams, file transfer, etc. Formal languages such as SDL [Z100], Z [Spiv89] may be used in addition to GDMO/GRM and IDL in the information and computational viewpoints in order to specify object behaviour.

The engineering viewpoint maps the computational objects to engineering objects supported by the ODP platform, introduces deployment concepts and supports the distribution transparencies. Computational objects are mapped to Basic Engineering Objects (BEOs). A remote BEO is represented to a local BEO through a *stub* object, which provides access transparency through functions known as *marshalling* and *unmarshalling*; these are equivalent to OSI presentation layer functions. A *binder* object maintains the binding between BEOs and can be thought as analogous to an OSI association between application layer entities. Finally, the stub and binder objects are supported by a *protocol* object which encapsulates the underlying protocol stack. Related BEOs may be grouped together in *clusters*, which become the unit of distribution i.e. they can be deployed and migrated together. Clusters may be grouped together in *capsules*, which typically map onto operating system processes. Capsules are deployed in *nodes*, which represent computing equipment attached to the network e.g. a general purpose workstation, a piece of switching equipment, etc. Finally, nodes contain the *nucleus* which realises the basic ODP platform environment.

4.2.5 OSI-SM / TMN and the ODP Viewpoints

Since the early days of the ODP framework, there have been efforts trying to describe OSI-SM in ODP terms. [Proct92] provides an analysis of OSI-SM from an ODP perspective, parts of which were later added as an example to the ODP recommendations. More recently, the Open Distributed Management Architecture (ODMA) [X703] does the same but in a more detailed manner and from the point of view of the OSI-SM recommendations. Various other researchers have addressed the issue of OSI-SM in the context of ODP. While we are going to describe relevant research and present our view in detail later on in this chapter, we consider briefly the relevant issues below.

There are two basic approaches for describing OSI-SM / TMN in ODP terms, which can be thought as being at the two ends of the relevant spectrum:

- every OSI-SM managed object becomes an ODP computational object, with manager objects interacting directly with them; or
- every OSI-SM / TMN agent becomes an ODP computational object, providing access to the contained managed objects which can be thought as ODP information objects mapped directly to engineering objects.

In the first case, the computational interface of every managed object is derived by mapping its GDMO specification to IDL. In this case, ODP mechanisms are used for object discovery, e.g. name servers and traders, and event dissemination, e.g. ODP event servers. In the second case, the computational interface of the agent is produced by mapping the CMIS specification to IDL. In this case, the OSI-SM mechanisms are used for object discovery i.e. scoping / filtering, and event dissemination i.e. event forwarding discriminators. In both cases, the communications mechanism can be either the Q_3 protocol stack, assuming that the ODP platform is ported over CMIS/P, or the protocol stack prescribed by ODP.

The OSI-SM/TMN environment described in Chapter 3, though not conforming to the letter of the ODP recommendations, can be thought as a distributed management software platform in the "spirit" of ODP. It actually uses the second approach described above but includes aspects of the first approach as follows. The distributed interactions follow the second approach, with manager-agent exchanges take place across Q_3 interfaces for TMN compliance. On the other hand, manager and managed objects are presented with APIs similar to those of ODP-influenced platforms. Manager objects can access either individual managed objects through the SMIB API, in a similar fashion to the first approach above, or whole agents through the RMIB API, in a similar fashion to the second approach. In fact, information objects are accessed through APIs as if they were separate computational objects. We will investigate the issues of mapping the OSI-SM / TMN model onto ODP-influenced distributed object frameworks in more detail in the rest of this chapter.

We will finalise this introduction to ODP with a comment on its use as a methodology for designing and specifying distributed systems. While the ODP viewpoints present an excellent set of tools for documenting the design of a complex distributed system, they are just that - a set of documentation tools. There is no clear way to go from one viewpoint to another, and the relationships between the information and computational views are not fully addressed. The word methodology implies a well-defined approach for system decomposition and component

identification. A methodology for TMN interface specification was discussed in section 2.3.2.4 of Chapter 2 [M3020][Gri96a], explaining how the designer starts from management service definition, moves into functional decomposition and then recomposition in order to arrive at a computational view. Such aspects are currently lacking in ODP. In fact, some refer to ODP as a *meta-methodology*, which needs to be specialised for a particular problem domain.

4.3 ODP-based Technologies

In this section we consider the Advanced Networked Systems Architecture (ANSA), the OSF Distributed Computing and Management Environment (DCE/DME) and the OMG Common Object Request Broker Architecture (CORBA) as ODP-based technologies. We concentrate more in OMG CORBA, since it will be considered as the base technology for the TMN in section 4.4. This presentation serves as a state-of-the-art description but also goes further, discussing issues behind the use of those technologies in TMN environments. In addition, the presentation of OMG CORBA addresses aspects that are not typically discussed in the relevant literature.

4.3.1 The Advanced Networked Systems Architecture

The Advanced Networked Systems Architecture (ANSA) [ANSA89a] was the first distributed software platform. It took further the concept of RPC-based software infrastructures compared to previous efforts by Sun Microsystems [Sun87][Sun88] and UCL [Wilb87]. ANSA was more than a distributed software platform, influencing the development of the whole ODP framework. In this section, we examine ANSA mainly as a distributed software platform and consider its suitability for TMN environments.

The ANSA platform, known as ANSAware, uses its own remote procedure call protocol. Object interfaces are specified in the ANSA IDL. Multiple objects can be organised into capsules, which map onto operating system processes. Objects in different nodes interact through the Remote Execution Protocol (REX), which relies on the underlying RPC protocol. Both synchronous and asynchronous remote execution mechanisms are supported. Remote operations are invoked through pre-processor extensions to the underlying programming language, which are specified in the Distributed Processing Language (DPL). The ANSAware environment and application APIs are in the C programming language.

Let’s examine now ANSA against the desired properties of object-oriented distributed frameworks defined in section 3.2.2 of Chapter 3. Though ANSA claims to be object-oriented, it is in fact object-based. Its IDL does not support inheritance and polymorphism while the relevant APIs are in the C programming language. Though Modula 3 and C++ language mappings were promised, they were never officially released. Remote operations are invoked by clients in a “pseudo” object-oriented fashion, through DPL directives i.e. real proxy objects do not exist in the local address space in a similar fashion to CORBA or to the OSIMIS RMIB/SMIB. Finally, there is no dynamic invocation interface.

The author considered porting the OSIMIS environment over ANSA during the first year of the RACE ICM project i.e. in 1992. The main reason was the increasing interest in ODP and relevant technologies in European telecommunications research projects. In addition, OSIMIS did not support at the time location transparency, so the ANSA trader could be used for this purpose.

This mapping could follow one of the two approaches described in section 4.2:

- every managed object would become an ANSA object, with its own IDL interface resulting from the GDMO to ANSA IDL mapping; or
- the whole agent would become an ANSA object, with an IDL interface mirroring the functionality of CMIS.

The first approach poses the problem that GDMO is object-oriented, supporting inheritance and polymorphism, while the ANSA IDL is not. It is of course possible to turn an inheritance hierarchy into a flat structure by repeating the inherited functionality. Such an approach though is a "hack" and does not support reusability and extensibility. In addition, if ANSA was adopted event dissemination facilities would need to be provided. With the OSIMIS infrastructure in place, including scoping, filtering, fine grain event reporting, logging and high-level object-oriented APIs in C++, this approach seemed to be more than one step backwards and was rejected.

The second approach means that the OSIMIS generic agent and manager infrastructures could be retained, including name resolution, scoping, filtering and event dissemination. The whole agent would become an ANSA object, with the manager-agent communications based on some form of "Q_3" interface over the ANSA protocols. Management applications would discover each other through the ANSA trader.

This approach seemed more promising and the author started investigating it but he soon faced a brick wall. Realising the CMIS service in IDL requires support for an *any* type, with semantics similar to the ASN.1 ANY, and the ANSA IDL did not support such a type. In addition, some CMIS operation parameters, e.g. filters, are "recursive" ASN.1 structures which cannot be expressed in ANSA IDL. CMIS is by definition an asynchronous service. While ANSA claimed to support asynchronous operation invocations through a "voucher" mechanism, the latter did not seem to work in the ANSA version at the time. ANSA provided a threads mechanism to be used with synchronous operations. This operated in user space and made distributed programs difficult, if not impossible to debug. As such, this approach was never pursued any further. A variation of this approach was pursued though by the author a few years later using CORBA, the

latter being the first ODP-based technology that had come of age. The CORBA-based approach will be described in detail in section 4.4.

It should be clear from the above discussion that ANSA represented a fairly early view of an ODP-based infrastructure and had a number of limitations. Due to those limitations, it is not possible to use ANSA as base technology for TMN systems. On the other hand, ANSA paved the way to more mature technologies such as CORBA and provided both the theoretical foundation and a first concrete implementation as a proof of concept for the ODP framework.

We will close this discussion on ANSA with a final remark. While ANSA claimed to be an *open* architecture, providing a solution for *open* distributed processing, this claimed openness is debatable. ANSA used its *own* protocols, despite the fact there was a lot of talk about substituting those with agreed protocols. In addition, there was no attempt to standardise the relevant APIs. There has always been only *one* ANSA product by Architecture Projects Management (APM) Ltd, the company formed to develop and promote ANSA. ANSA applications can only interwork with other ANSA applications on the same platform. All these aspects point to a *closed* rather than *open* solution. The author expressed those views at the time. [Marc95] makes exactly the same remarks for the OSF DCE/DME, which we will examine in the next section.

4.3.2 The OSF Distributed Computing and Management Environment

The Open Software Foundation (OSF) Distributed Computing Environment (DCE) [DCE] is an integrated collection of a support environment, tools and services for the development, deployment and maintenance of distributed applications. Its core components include the DCE RPC protocol and programming environment; a threads facility; the DCE Directory Services, including the Cell Directory Service (CDS) used for naming and location transparency; the DCE Security Service which provides Kerberos-based authentication, stream integrity and access control; and the DCE Distributed Time Service (DTS). The idea of the DCE was to provide a vendor-neutral platform for building distributed applications, building on work done by various OSF vendor members and reusing their software infrastructure. In this section we consider briefly DCE as an a potential environment for distributed TMN applications.

DCE object interfaces are specified in the DCE IDL, which is different to the ANSA IDL. Multiple objects can be organised into DCE *servers*, which map onto operating system processes. Communication between those servers is supported by the DCE RPC protocol. The DCE IDL does not support inheritance and polymorphism, while the relevant APIs are in the C

programming language. DCE is a procedural environment instead of being object-based or object-oriented. Clients call directly procedures of interfaces, without a proxy object in the CORBA/OSIMIS sense, or even a "pseudo" object handle, as in the ANSA DPL. In other words, DCE is simply an RPC environment, with additional support services for naming and security. It would be rather stretched to call it an ODP-based environment. The reason we consider it here is that it was a commercial attempt towards a distributed software platform that was fairly popular before the advent of fully distributed object technologies such as OMG CORBA and MS DCOM.

The same issues regarding the use of ANSA in TMN environments hold also for DCE. It is not possible to map directly the OSI-SM/TMN model onto DCE using any of the two approaches described above: the problems described for the ANSA IDL hold also for the DCE IDL. The reader may remark that it is possible to use a basic mechanism, such as the DCE IDL and RPC, and build on top of them a fully object-oriented distributed framework. For example, MS DCOM uses DCE while [Autr94] describes how to map the CORBA IDL onto the DCE IDL. While this is possible, we examine here the possibility of using DCE *as is* rather than using it as a starting point for developing a whole new framework. In summary, DCE is *not* suitable technology for TMN systems.

While DCE targeted general purpose distributed systems, an OSF initiative known as the Distributed Management Environment (DME) [DME] tried to build on it and provide a distributed object-oriented management framework. DME tried to combine every known management technology into one integrated framework. It uses DCE for the communications between management applications but it also uses Q_3 and SNMP for accessing managed elements with such interfaces. While the DCE is hidden underneath the DME, applications are built using the CORBA framework and APIs which are in turn mapped to the DCE IDL and RPC. Adapter objects were supposed to convert between the IDL/RPC framework and CMIS/P or SNMP. A fine grain event service would be provided to applications while the software platform would include a generic graphical user interface, similar to a MIB browser [Pav92a].

The DME idea originated from the fact that DCE provided an interoperable communications environment with naming services and security, while CORBA provided the object-oriented APIs but not an interoperable protocol at the time. NMF and X/Open had just started their Joint Inter-Domain Management (JIDM) task force to investigate the mapping of GDMO and the SNMP SMI to CORBA IDL, so it was thought easy to provide adapters to Q_3 and SNMP. The idea was to reuse existing software components from Hewlett-Packard, IBM and Tivoli. It is not surprising that the DME "dream" never materialised. A discussion on the reasons behind DME's failure can

be found in [Marc95], which is titled "Icaros, Alice and the OSF DME". The title alludes that DME tried to fly "too close to the sun", as Icaros in ancient Greek mythology. It also points to Lewis Carroll's Alice in Wonderland!

While the DME approach seemed unrealistic at the time, relevant research since then has made possible the combination of protocol-based TMN solutions and new generations of distributed object technologies such as CORBA and DCOM. Related work and the author's approach will be presented in section 4.4.

4.3.3 The OMG Common Object Request Broker Architecture

4.3.3.1 The Object Request Broker Model

While the ISO/ITU-T ODP [ODP] is a theoretical framework for specifying and building distributed systems, the Common Object Request Broker Architecture (CORBA) [CORBA] can be seen as its pragmatic counterpart. The relevant architecture and specifications have been the result of work by the Object Management Group (OMG) and represent an industrial approach towards open distributed systems. A number of vendors offer CORBA platforms that conform to the relevant specifications. These provide both "horizontal" openness through standard APIs, resulting in application portability across platforms, and "vertical" openness through agreed protocols, resulting in application interoperability across platforms.

The OMG CORBA model is shown in Figure 4-5. The relevant paradigm is a client-server one, with distribution transparencies provided through the Object Request Broker (ORB), which has similar functionality to the ODP platform. The unit of distribution is the single object as opposed to the OSI-SM object cluster. Objects in client and server roles communicate through the ORB. Special server objects provide generic Common Object Services [COSS], such as naming, event serving, trading, etc. Interoperability is achieved through the formal specification of server interfaces in the CORBA Interface Definition Language (IDL) and through the underlying protocols. Portability is achieved through standard APIs for objects in client and server roles. These map the CORBA IDL onto various programming languages, such as C, C++, Smalltalk, Java, etc. The CORBA object model, i.e. the IDL language which defines the properties of interfaces and subsequently objects, and the relevant APIs have been addressed first, while the underlying protocols may be replaced.

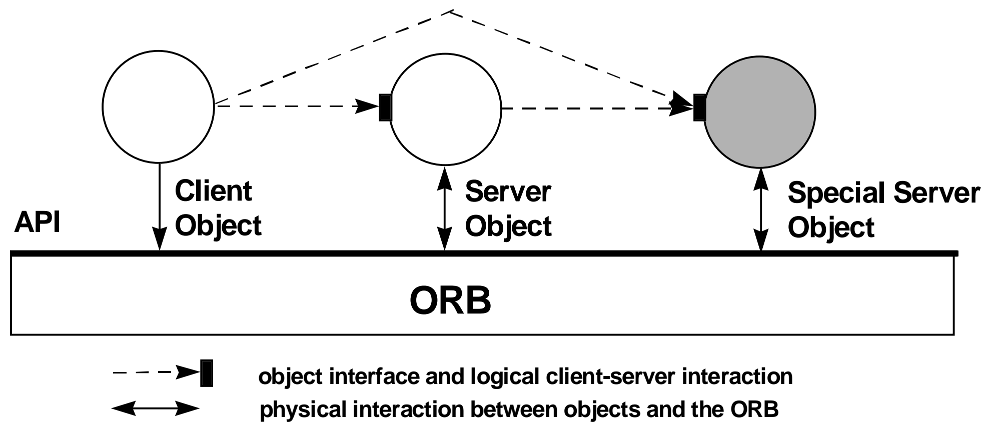


Figure 4-5 The OMG CORBA Model

CORBA objects can be organised into "containers" similar to the ANSA/ODP capsules, which are called *servers* and map onto operating system processes. Note that this use of the term server is different to the same term referring to an object in server role: CORBA objects in client and server roles may be organised into servers i.e. operating system processes. Objects within a server may communicate either directly, exploiting the knowledge they are co-located within that server, or indirectly through the ORB. A CORBA server is in fact equivalent to both the ODP concepts of capsule and cluster. In other words, there is no notion of cluster in CORBA but only the notion of capsule. The ORB itself is a special daemon that runs in every CORBA-capable network node. Note that the ORB as depicted in Figure 4-5 models the union of all those ORB daemons running on different network nodes. This conceptual distributed ORB together with the various generic servers that offer special services is also referred to as the Distributed Processing Environment (DPE); the latter is terminology originated in TINA [TINA].

Since CORBA is an object-oriented, mature, ODP-influenced technology, it can be considered as the basis of TMN systems in lieu of the OSI-SM and Directory. As such, we will consider it here at the same level of detail as we did for OSI-SM in section 2.2 of Chapter 2, covering both its information modelling and distributed object access aspects.

It should be noted that OMG CORBA is not the only emerging distributed object technology: Microsoft's Distributed Component Object Model (DCOM) [DCOM] is another similar technology. The key difference between CORBA and DCOM is that the former is based on collectively produced, openly available specifications by the OMG, while the latter is the work of a single company. Any vendor may develop a CORBA-compliant platform and compete in the relevant marketplace while there exists only one DCOM platform, produced by Microsoft. This is one of the key reasons we consider CORBA instead of DCOM in this thesis. Despite that, and

given the similarities between CORBA and DCOM, we will refer specifically to any differences between the two during the detailed presentation of CORBA.

4.3.3.2 The Information Model

The CORBA information model is "indirectly" defined by the Interface Definition Language (IDL), which defines the properties of object interfaces and subsequently objects. In the strict ODP sense, IDL is a computational viewpoint language since it defines the interactions among objects through their interfaces. On the other hand, in the absence of any formal information modelling language that is specific to CORBA, we can also treat IDL as an information modelling language.

The CORBA information model is fully object-oriented, in a similar fashion to OSI-SM. Objects are characterised by the *interfaces* they support. While an ODP object may support multiple interfaces bound to a common state, the current OMG IDL allows only a single interface per object. It should be noted here that the DCOM IDL allows multiple interfaces per object. In fact, the OMG model defines objects through the specification of the relevant interfaces. As such, there is no direct concept of an object class. Object interfaces may be specialised through inheritance while multiple inheritance is also allowed. The root interface in the inheritance hierarchy is of type *Object*. The IDL specification technique is more monolithic than the GDMO piecemeal approach: the minimum re-usable specification entity is the interface definition as opposed to the individual package, attribute, action and notification in GDMO. IDL may be regarded as broadly equivalent to the GDMO/ASN.1 combination in OSI management, though less powerful and with some important differences highlighted below.

An OMG object may accept operations at the object boundary, have attributes and exhibit behaviour. Such an object is used to implement a computational construct. In a management context, an object may behave as a manageable entity, modelling an underlying resource.

Objects accept object-oriented operations, similar to the GDMO actions. Exceptions may be defined as part of an interface and be associated to operations. The latter take *in* and *out* arguments, which may be of arbitrary IDL types. Objects may also have attributes of arbitrary IDL types. Attributes accept *get* and *set* operations while those specified as *readonly* accept only *get* operations. Attributes can not be specified with "set by default", "add" or "remove" properties as in GDMO. In addition, only standard exceptions may signify an error during the get and set operations. This is in contrast to GDMO, where arbitrary class-specific errors may be defined to model exceptions for attribute operations. Defining an attribute as part of an object

interface results simply in methods named *<attr>_get* and *<attr>_set* to be part of that interface. The limitations discussed above may be lifted if, instead of defining attributes, an information model designer defines directly associated access methods. In this case, *<attr>_setToDefault*, *<attr>_add* and *<attr>_remove* methods could also be defined, while all the attribute access methods may have associated exceptions.

A key difference between OMG and GDMO objects is that the former do not allow for the late binding of functionality to interfaces through optional constructs similar to the GDMO conditional packages. An OMG object type is an absolute indication of the characteristics of an instance of that type. An additional major difference is that in IDL it is not possible to specify event types *generated* by an object: events are modelled as "operations in the opposite direction". As such, events are specified through operations on the interface of the receiving object. An OMG managed object needs to specify a separate interface containing all the events it can generate; the latter needs to be supported by managing objects that want to receive those events. This peer-to-peer modelling of events is shown in Figure 4-6. The interface A models the operations supported by object A while the interface B models the events emitted by object A and received by object B. This split in the specification of operations and events is one key drawback of the model. There are more differences with respect to the way events are disseminated, but these are discussed in the next section, since they are related to the access paradigm.

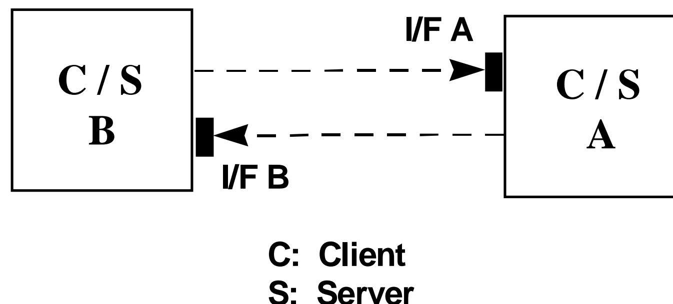


Figure 4-6 Peer-to-Peer Object Interaction for Events and Asynchronous Operations

OMG objects do not provide a built-in operation for instantiation of interfaces by client or managing objects. The reason for that is that OMG takes a "programmatic" view of object interfaces and, as such, a create operation is meaningless before that interface exists. While GDMO objects appear to accept create operations, these are essentially targeted to the agent administering them. As such, interface creation in OMG may only be supported by existing interfaces: *factory* objects may be defined that allow client objects to create application specific

interfaces. This approach is not flexible since a specific factory interface is necessary for every interface that can be dynamically created.

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

While the OMG IDL object model has many similarities to GDMO, a marked difference concerns naming. OMG objects can be identified and accessed through *Object References*. The latter are assigned to objects at creation time and are opaque types, i.e. they have no internal structure and, as such, do not reveal any information about the object. Object references are in fact similar to programming language pointers and are used to de-reference a “proxy” server object within a client’s address space. While distributed objects are effectively accessed through references, an object may also be assigned one or more names. The latter are distinct from objects, unlike OSI-SM where an object always has a name. Actually OMG objects need not have names at all as they may accessed through their interface reference. In addition, names may be assigned to objects but this mapping may change at any time. Names are assigned to objects through the Name Service [COSS], which provides a directed graph of naming contexts with potentially many roots. A point in the graph may be reached via many routes, which means that an object may have many names. This is in contrast to OSI-SM where there exists a naming tree instead of a naming graph and objects have exactly one name. The name service may be used to assign names to objects and to resolve names to object references.

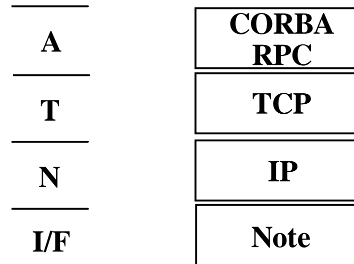
Finally, while the CORBA IDL supports inheritance, it does not support polymorphism in the same fashion as in object-oriented programming languages or in GDMO: it is *not* possible to redefine an operation or attribute in a derived interface and invoke its behaviour through a reference to the parent interface while this is possible in GDMO. This is a particularly important

limitation which shows that inheritance in CORBA IDL is simply a means of "bundling" properties together rather than using it for designing generic, polymorphic distributed systems. In a related fashion, there is no support for allomorphism.

4.3.3.3 The Access Paradigm

OMG CORBA was designed as a distributed software infrastructure in which the access protocol is secondary compared to the underlying APIs, which represent agreed IDL mappings to programming languages. Of course, an agreed protocol is necessary in order to achieve interoperability between CORBA platforms of different vendors. The OMG 1.x versions of CORBA specification left completely open the choice of access protocol and concentrated only on programming language bindings. Version 2.0 specifies a Remote Procedure Call (RPC) protocol, known as the General Inter-Operability Protocol (GIOP) [GIOP]. Two different transport mappings have been currently defined for the latter, the Internet Inter-Operability Protocol (IIOP) [IIOP] over the Internet TCP/IP (shown in Figure 4-7) and the DCE Common Inter-Operability Protocol (D-CIOP). IIOP has been by far the most widely deployed open protocol suite, in fact few commercial CORBA platforms support the D-CIOP. The rest of the CORBA software infrastructure is relatively independent of the underlying protocol, which could change without any effect on the APIs and the application software.

The GIOP/IIOP is a connection-oriented reliable RPC protocol that uses TCP and IP as transport and network protocols respectively. Applications that use CORBA-based communications are guaranteed transport reliability, in a similar fashion to OSI-SM. The RPC protocol is a general *request/response* protocol in which the exact content of the request and response packets is governed by the IDL specification of the accessed CORBA interface, in a similar fashion to the CMIS/P and GDMO coupling. The access service / protocol offers no special facilities for object discovery, similar to the OSI-SM scoping and filtering. Such facilities are provided instead by naming servers and traders. In summary, the CORBA RPC protocol provides a *single* object access mechanism, with higher-level facilities provided by standard OMG servers [COSS].



Note: The Internet interface layer may comprise many different protocols

Figure 4-7 The CORBA IIOP Protocol Stack

Objects in client roles interact with objects in server roles through "proxy" objects available in the local address space. An object reference is required in order to be able to get access to the proxy object representing a remote object in server role and invoke its methods. This reference can be obtained through the ORB by "binding" to the remote object by type, if the host where the object is running is known. The problem with this approach is that it violates location transparency. The correct approach which supports location transparency is to discover an object's reference either by name, through the name server, or by type and potentially other properties through the trader. A naming server [COSS] can be used to map one or more names to an interface reference. When an object instance is created, the naming server needs to be "notified" of the mapping between the object's name and its interface reference. Subsequently, client objects can resolve object names to object references through the naming server. A trader [X9tr] supports more sophisticated queries, matching static properties and dynamic attribute values of the target object(s), as it was described in section 4.2.3.

Having obtained a reference to an object, operations are invoked through the associated proxy object in a *synchronous* fashion only. The only possibility to perform operations asynchronously is to define them as *oneway* in the IDL specification. "One way" operations cannot have associated output arguments and exceptions. When defining one way operations, a designer needs to define as well the symmetric operations which model the relevant results and errors; these will be part of the client's interface, with the client and server roles reversed for the forwarding of the results / errors. The object reference of the client is typically passed as an argument in the "master" one way operation so that the server can "return" the results / errors. The peer-to-peer interaction for asynchronous operations was depicted in Figure 4-6. The interface A is "half" of what it should normally be, while the interface B is the other "half" of A, modelling the results and errors for the operations to A.

This approach to asynchronous operations is hardly satisfactory. Operations need to be defined as both oneway and non-oneway, with the results / errors requiring a separate interface. This increases considerably the complexity for asynchronous operations. In addition, the whole approach is somehow a "hack": asynchronicity should be a property of the API rather than the IDL interface via the oneway mechanism. For example, the OSIMIS RMIB/SMIB APIs described in the section 3.4 of Chapter 3 support both synchronous and asynchronous modes for invoking operations on remote GDMO objects. Given the significance of an asynchronous mode of execution in single-threaded environments, OMG intends to address asynchronous operations properly in CORBA version 3.0 [Vino97].

Operations invoked on proxy objects are strongly-typed and require the existence of static, precompiled knowledge of the target IDL interface in the client's program, in a similar fashion to the static OSIMIS SMIB model. CORBA supports also a Dynamic Invocation Interface (DII) which overcomes this limitation. The DII is a dynamic, weakly-typed interface that uses a generic operation interface. Operation types are specified in string form while the arguments, results and exception parameters are specified as IDL *any* types. This is similar to the OSIMIS RMIB and dynamic SMIB models. The DII supports also an asynchronous mode of operation which does not require operations to be defined as one way operations; this is in fact known as the *deferred synchronous operation* model. The DII could be used for generic applications such as MIB browsers etc. It should be noted though that not all commercial CORBA platforms support the DII. In fact, the author has not experienced any use of the DII in the various European research projects which have been using CORBA.

Finally, notifications in ODP/OMG are supported by *event* channels. Emitting and recipient objects need to register with those channels which are created and managed for every type of notification. Emitting objects invoke an operation on the relevant event channel while the notification is passed to registered recipient objects either by invoking operations on them (*push* model) or through an operation invoked by the recipient object (*pull* model). The interaction depicted in Figure 4-6 reflects the push model, while the event channel object is not shown. Event channels can be in principle federated for scalability and inter-domain operation but the relevant issues have not yet been addressed. The key difference with OSI-SM is the lack of fine grain filtering, which results in less power and expressiveness and more management traffic. OMG is currently working towards the specification of *notification* servers which will provide filtering and will also undertake the management of channels, providing a higher-level way to deal with notifications.

4.3.3.4 Summary

It should be clear from the above presentation that though CORBA is not as powerful as OSI-SM, it is in an ODP-influenced technology that has come of age and has a number of the attributes sought for supporting distributed network and service management systems. We will finalise this section by examining CORBA against the desired properties of object-oriented distributed frameworks, as defined in section 3.1.2 of Chapter 3.

CORBA projects a fully object-oriented framework since its IDL supports inheritance while there exist standard mappings to object-oriented programming languages. Its APIs are user-friendly, hiding communication and protocol details through "proxy" objects that represent remote objects as if they were in the same address space. They are in fact similar to the OSIMIS SMIB [Pav94b] and the NMF GDMO/C++ [Chat97] models that were discussed in Chapter 3, though they do not support information caching mechanisms. CORBA supports a Dynamic Invocation Interface (DII) which does not require pre-compiled knowledge of remote IDL interfaces in clients, providing for a dynamic, weakly-typed style of interaction suitable for generic applications. CORBA is open both in terms of standard APIs and communications protocols. Finally, CORBA supports access transparency through the IDL and the ORB APIs. It also supports location transparency through the naming server and trader, which are special servers.

4.3.4 A Summary of the Relevant Technologies

Having presented ANSA, the OSF DCE and OMG CORBA as ODP-based technologies, we will finalise this section with a summary and comparison of the characteristics of those technologies in relation to the desired properties of object-oriented distributed frameworks. We will also add as a fourth technology the combination of OSI-SM together with its realisation through the OSIMIS platform. Though OSI-SM / OSIMIS are not, strictly speaking, ODP-based, we have shown in this thesis that they exhibit a number of aspects of ODP systems and, as such, we can consider them in this summary and comparison.

It should be emphasised again that OSI-SM is a compositional model, based on the "object cluster" principle, so it cannot be used instead of any of those other technologies (or rather it can, but this is a degenerate case). On the other hand, any of the other technologies could be considered instead of OSI-SM. Table 4-1 shows the relevant technologies and their properties which are summarised below.

ANSA's IDL is not object-oriented while remote objects are accessed through DPL pre-processor extensions, without true proxy objects. There have not been mappings to C++, Smalltalk or Java

but there has been an experimental mapping to Modula 3. DCE's IDL is not object-oriented, the APIs are procedural while there exists only a mapping to C. Both ANSA and DCE do not support a dynamic invocation interface and the *any* type in their IDL.

CORBA's IDL is object-oriented but it does not support true polymorphism through the redefinition of operations and attributes in derived interfaces. The APIs are fully object-oriented and there exist multiple language mappings. A dynamic invocation interface and the any type are also supported.

OSI-SM's GDMO is fully object-oriented and supports inheritance and polymorphism. The OSIMIS APIs are fully object-oriented and support both static and dynamic invocation paradigms, in the latter case through the mapping of the ANY DEFINED BY ASN.1 construct to C++. A manager/client mapping to Tcl has been presented while there exist similar Java mappings in other TMN platforms. On the other hand, the major mapping both in OSIMIS and in the NMF TMN/C++ APIs is in C++.

Technology and properties	O-O Abstract Language	O-O Language Mapping	Multiple Language Mappings	Dynamic Invocation Interface
ANSA	no	no	partly	no
DCE	no	no	no	no
CORBA	yes/partly	yes	yes	yes
OSI-SM / OSIMIS	yes	yes	partly	yes

Table 4-1 A Comparison of Distributed Object Technologies

It is evident from the Table 4-1 that ANSA and DCE fail to satisfy important properties of object-oriented distributed frameworks. As such, they cannot be used as base technologies for the TMN for reasons discussed in detail in sections 4.3.1 and 4.3.2. It is also evident from the Table 4-1 that CORBA is an ODP-influenced technology that has come of age, since it satisfies almost all those properties. It lacks though full support for polymorphism, proper support for asynchronous APIs and event-reporting based on filtering.

4.4 Using Distributed Object Technologies in TMN

According to the analysis in the previous section, OMG CORBA is the first ODP-influenced technology that satisfies most of the requirements for building object-oriented distributed systems with dynamic properties. In this section, we will examine in detail the issues behind using CORBA and ODP principles in TMN environments. The key issue is to investigate exactly how OSI-SM can be replaced by CORBA as the base technology for the TMN. We need thus to investigate in detail the relationship between OSI-SM and CORBA or, more generally, the relationship between OSI-SM and ODP.

4.4.1 Mapping OSI-SM to ODP

In section 4.2.5, we discussed in a "flash forward" fashion two approaches for describing the OSI-SM model in ODP terms. In the first one, every managed object becomes a separate ODP computational object, with no OSI agent functionality. In the second one, every agent becomes an ODP computational object, containing managed objects as information objects. These approaches can be thought as being at the ends of the spectrum, while mixed approaches are also possible. The first approach has its origins in the ODP community while the second one was conceived by the author and other researchers trying to preserve the benefits of the OSI-SM model in ODP environments. We present the first approach below.

Since the early days of ODP, there have been various attempts to describe OSI-SM in ODP terms. [Proct92] was the first attempt, part of which found its way into the ODP-RM [ODP] as an appendix. The approach taken was to consider managed and managing objects as ODP objects, communicating via supporting ODP platform mechanisms. This implies that the functionality of OSI agents, such as name resolution, scoping, filtering, object creation and access control, is not explicitly present. The functionality of mapping object names to interface references and object creation need to be supported through ODP mechanisms. [Proct92] does not address those aspects but suggests that OSI-SM mechanisms should be used for event reporting through EFD support managed objects.

A similar approach has been more recently studied in the Open Distributed Management Architecture (ODMA) [X703], which is the ISO/ITU-T approach to describe OSI-SM in ODP terms. ODMA tries to provide a more general framework that can be mapped to either ODP-based object technologies, such as OMG CORBA, or to OSI-SM communication protocols and relevant engineering concepts. OSI managed and managing objects map onto ODP objects and

interfaces. The ODP trader is used for discovering interface references, according to desired object properties. Object creation is supported through factory interfaces, which can be also discovered through the trader.

In the case of an ODP-based supporting technology, managing and managed objects communicate directly with each other over the ODP platform. When the underlying supporting technology is OSI-SM-based, the OSI agent becomes an "operation dispatcher" in the engineering viewpoint which performs operations to managed objects. It also becomes a "notification dispatcher" that forwards notifications to managing systems. One or more notification dispatchers may be also necessary in a managing system in order to deliver the notifications to the requesting managing objects. This is *exactly* the functionality provided by the MIBAgent object in OSIMIS agent applications and by the RMIBAgent objects in manager applications. This means that the object-oriented decomposition presented in Chapter 3 is in accordance with the engineering decomposition of OSI-SM systems in ODP terms, as suggested in [X703].

This ODP view of OSI-SM implies that the resulting framework does not support scoping, filtering and multiple operations to managed objects. In addition, if CORBA is used as the underlying platform, it is mentioned that notifications should be disseminated using relevant mechanisms i.e. OMG event servers and channels. When OSI-SM based platforms are used, the relevant protocols and supporting engineering concepts such as agents and notification dispatchers should be hidden behind the ODP platform APIs. The intention is to allow for the specification of management systems from an information and computational perspective in an implementation neutral fashion. The use of an OSI-SM or ODP-based technology is considered an engineering viewpoint decision that does not affect the system specification.

We could characterise the above approach as a "least common denominator" one, in which the OSI-SM framework is "pruned" to fit the ODP model. Despite its ODP orientation, [X703] recognises the fact that multiple object access through scoping and filtering and event dissemination through event filtering and EFDs may need to be exposed in the computational viewpoint. This leaves open the possibility for other potential mappings between OSI-SM and ODP. We will present such an approach in the rest of this chapter.

4.4.2 Mapping OSI-SM GDMO to CORBA IDL

The previous approach for describing OSI-SM in ODP terms assumes that it is possible to map a managed object specified as a GDMO information object to an ODP computational interface. In the absence of an ODP computational modelling language, we will assume that this language is the CORBA IDL and we will consider the issues of mapping GDMO to CORBA IDL. This is in accordance with the fact that OMG CORBA is considered as the pragmatic counterpart of ODP and the fact that OMG specifications may evolve into relevant ODP recommendations.

The issue of mappings between GDMO and CORBA IDL has been addressed by the Joint Inter-Domain Management (JIDM) task force between the NMF and X/Open. Though the main intention behind this work was to result in the specification of generic gateways between different management technologies, the same principles and mappings can be used to support pure CORBA-based management systems. The JIDM work started in 1993 and the first important outcome was the comparison of the Internet SNMP, OSI-SM and OMG CORBA object models described in [Rutt94]. The *specification translation* aspects followed [JIDM95], including the generic mapping of GDMO to CORBA IDL. Since then, the focus has been the dynamic *interaction translation*, which can be used to support the construction of generic application-level gateways. We discussed "indirectly" some of the aspects behind mapping GDMO to IDL in section 4.3.3.2, while discussing the CORBA information model. We discuss again and elaborate on those aspects below.

While IDL interfaces have attributes in a similar fashion to GDMO objects, it is not possible to map GDMO to IDL attributes directly. This is because IDL attributes have only *get* and *set* properties, while GDMO attributes have additional *setToDefault*, *add* and *remove* properties. In addition, it is not possible to define specific exceptions associated with access to attributes in IDL, while this is possible in GDMO. As such, GDMO attributes should map to access methods in accordance with the relevant properties e.g. <attr>_get, <attr>_set, <attr>_setToDefault, <attr>_add and <attr>_remove.

While this approach solves partly the mismatch problem between GDMO and IDL attributes, it creates other problems. For example, it is not possible to interrogate the CORBA interface repository about the attributes an interface has in order to access those attributes through the dynamic invocation interface; the latter is useful for generic applications such as MIB browsers. In addition, attributes cannot be exported to the trader in the usual way, through the attribute name and type. GDMO attributes also exhibit "MATCHES FOR" properties which are used for filtering. There is no equivalent in IDL which means that filtering cannot be easily supported. We

will revisit the issue of filtering in ODP-based management environments later in this section. In summary, there are problems with respect to the mapping of GDMO attributes to equivalent IDL constructs, emanating from the superior expressiveness of GDMO. The only proper solution would be for OMG to reconsider the IDL attribute model and "upgrade" it to an equivalent of the GDMO one.

GDMO actions can be naturally mapped onto IDL methods with input argument the action information and output argument the action result. Action parameters, which signify action-specific errors, are mapped onto IDL exceptions. GDMO notifications can be mapped onto separate interfaces that should be supported by managing objects and event channels. Two separate interfaces should be generated for the notifications of a managed object class, one for the "push" and one for the "pull" model. Notifications should be typically mapped onto oneway operations, for non-confirmed notifications, and onto normal operations, for confirmed notifications. In general though, the key problem with the mapping of notifications is that they result in different interfaces, fragmenting the relevant specification. In summary, the CORBA support for notifications from a specification point of view is not satisfactory. In addition, the notification dissemination model is less powerful than the OSI-SM, as it was discussed in section 4.3.3.3.

An additional difference between GDMO and CORBA IDL concerns the dynamic binding of functionality to managed object instances through conditional packages. This is a key feature of GDMO, used very often by information model designers, while it is not supported in IDL. The only solution is to make the functionality of GDMO conditional packages "mandatory" from a specification point of view i.e. their attributes and actions will be part of the resulting IDL interface. Their presence though will become an implementation issue. CORBA supports a standard *not_implemented* exception which will be raised whenever a method of a non-supported package is invoked. An interface should "advertise" the supported conditional packages through the *packages* attribute of the *i_top* interface, which will result from the translation of the OSI-SM *top* class [X720].

This approach is all that can be done regarding the mapping of conditional packages and has a number of problems. First of all, the functionality of assigning packages dynamically to object instances at creation time is lost. A more important problem is that conditional packages are sometimes used by information model designers in a fashion that cannot be supported through "hardwired" implementations. For example, the monitorMetric class specified in [X739] uses conditional packages in such a way that their inclusion (or not) has an impact on the functionality

of the relevant object instance. While this is arguably poor modelling practice, such cases have often been used by information model designers. The only solution to this problem is to use non generic mappings but then generic application gateways between OSI-SM and CORBA cannot be supported. Another solution would be for ISO/ITU-T to "correct" any existing specifications that present this problem and instruct information modelling groups on the proper use of GDMO conditional packages, in an IDL-compatible fashion.

These are the rules for mapping GDMO to CORBA IDL, as specified by the JIDM group [JIDM95]. The author added his own views and opinions regarding various aspects of this mapping. While the latter goes a long way towards reconciling the two models, some semantics are inevitably lost in the translation. The most important problems concern the mapping of GDMO attributes to CORBA attributes, the mapping of notifications onto separate "unrelated" interfaces and the problem of conditional packages. In addition, CORBA does not allow the redefinition of attributes and operations in derived interfaces, so true polymorphism cannot be supported. This presents another mapping problem since this useful feature is often used in GDMO specifications. In the absence of true polymorphism, allomorphism is also meaningless in CORBA. We will ignore these problems at present, assuming that future CORBA versions may address the problem of a fully object-oriented IDL that supports polymorphic operations.

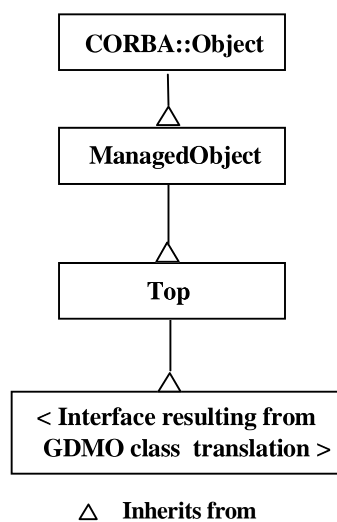


Figure 4-8 IDL Inheritance Hierarchy Resulting from GDMO Mapping

Having presented the rules for IDL to GDMO translation, it is possible to map OSI-SM GDMO managed objects to CORBA IDL interfaces and preserve all the work that has gone into the relevant OSI-SM / TMN specifications. The relevant translation may support gateways between CORBA and OSI-SM / TMN applications. It may be also used to support the native operation of

management systems entirely in CORBA, as will be investigated in the rest of this chapter. The equivalent IDL interfaces follow exactly the same inheritance lattice as in GDMO, while the *i_top* interface is equivalent to the OSI-SM *top* class [X720]. The i_top interface inherits from the *i_ManagedObject* one, which in turn inherits from CORBA's Object, as do all the IDL interfaces. The resulting inheritance hierarchy is depicted in Figure 4-8.

The i_ManagedObject interface may support functionality common to all the managed objects, such as getting an object's name, accessing a number of attributes with one operation, evaluating a filter and returning the interface references of its superior or of its subordinate objects in the containment hierarchy. The latter is similar to a part of the functionality of the MO class in OSIMIS, as described in Chapter 3.

```
interface i_top : i_ManagedObject
{
    string              objectClass_get ();
    string              nameBinding_get ();
    StringList_t        packages_get ();            // conditional
    StringList_t        allomorphs_get ();          // .. (but not used)
};

interface i_uxObj : i_top
{
    SimpleNameType_t  uxObjId_get ();
    UTCTime_t         sysTime_get ();
    string            wiseSaying_get ();
    void              wiseSaying_set (in string);
    string            wiseSaying_setDefault ();
    ObservedValue_t   nUsers_get ();
    void              echo (in string, out string);
};
```

Code 4-1 The Top and UxObj Managed Object IDL Interfaces

The Code 4-1 caption above shows the potential mapping of the OSI-SM top class [X721] and the uxObj class used in the examples of Chapter 3, following the rules presented above which are largely based on the JIDM approach. The translation of the uxObj class will also comprise a notification interface and a factory interface which are not shown above. The GDMO to IDL translation assumes that ASN.1 data types for attributes, actions, notifications and parameters (i.e. exceptions) are also translated to CORBA IDL. [JIDM95] specifies the rules for ASN.1 to IDL translation. A complete specification of those classes in both GDMO and CORBA IDL can be found in Appendix C.

4.4.3 An Initial Mapping of the OSI-SM Model to CORBA

Having discussed the mapping of GDMO to CORBA IDL, we will consider how CORBA can be used instead of OSI-SM as the basis for TMN systems. We will consider first a pure ODP approach, which uses trading for object discovery and disregards completely hierarchical names based on containment relationships. This is the approach advocated often by ODP enthusiasts but there has never been an attempt in the literature to consider the issues involved. We will thus try to investigate if and how this approach can be used to support telecommunications management systems. We will then present another, more pragmatic approach, which uses name services and hierarchical containment relationships instead of trading. The second approach will be extended further to include TMN aspects such as object clustering, scoping, filtering and EFD-based event dissemination, migrating essentially the OSI-SM / TMN framework over CORBA.

4.4.3.1 A ODP-Oriented Approach Using Discovery Through Trading

We will assume first that GDMO specifications are translated to CORBA IDL as described above. If CORBA is used as the underlying access and distribution mechanism, managed objects will be realised as CORBA objects. The key difference is that clusters of managed objects logically bound together, e.g. objects representing various aspects of a managed network element, are not now seen collectively through an agent. As such, an important issue is to provide object discovery facilities which in TMN environments are supported first through directory access and then through scoping and filtering. Such facilities are very important in management environments where many instances of the same object type typically exist, with names not known in advance, e.g. call objects.

We will assume that containment relationships are treated just as any other relationship and that they are not used for naming. An alternative naming scheme might exist but this is not important since the solution is based on trading rather than naming. The trader will provide facilities similar to filtering based on assertions on object properties and attributes. It should be mentioned that trading services are still under development by OMG. An early implementation of the ODP trader specifications using CORBA has been used in the ACTS VITAL project, provided as background information to the project by Alcatel Alsthom Recherche [Trate96]. This particular implementation supports only equality checking on attributes and properties. This functionality is inferior to the OSI-SM filtering but we will assume that traders will eventually support more sophisticated assertions.

In a distributed management environment, managing objects will discover managed objects based on assertions on properties and attributes through the trader. An important issue with the use of trading is that federation is a key aspect in order to achieve scaleable systems. In essence, it will be necessary to have dedicated traders for every logical cluster of managed objects, for example in every managed element, in order to reduce traffic and increase real-time response. Those "low-level" servers will need to be unified by "higher-level" servers in a hierarchical fashion. Federation issues become thus very important but have not yet been worked out and are not simple. Even with such facilities in place, the management traffic in terms of the required application-level packets will be at least double that of OSI-SM. In CORBA, matching object references will be returned by the trader to the client object and the operations will be performed on an object-by-object basis. In OSI-SM, a multiple object access request will be sent in one packet while the results will be returned in linked replies, one for each object accessed. The use of CORBA for network management through federated trading is depicted in Figure 4-9.

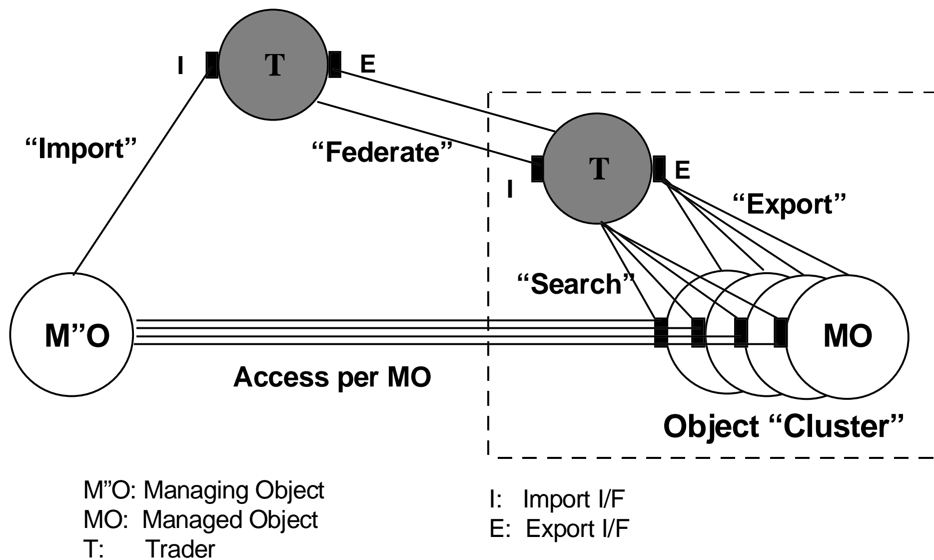


Figure 4-9 Object Discovery Through Trading

We will consider now the same example we considered in section 2.2.4 of Chapter 2 for OSI-SM in order to demonstrate in more detail the operation of this CORBA-based framework. The manager in this case will be a CORBA client object which will have to discover the routing table entries in a router that point to a particular "next hop address". through the trader. We will assume that there exists a trader in the router's node so that interactions between the router managed objects and that trader do not incur management traffic. This trader together with other traders will constitute a federated trader network, which has similarities to the global OSI Directory [X500]. The difference is that federation in the directory is based on hierarchical

information structuring principles while the global trading "service space" does not typically exhibit hierarchical properties and, as such, federation is much more difficult to provide.

Objects in the router will have to export their properties and attributes to the trader when they are created and withdraw them when they are deleted. The interface reference of the local trader is supposed to be a "well known" parameter. The local trader should tell "superior" or "neighbour" traders about the service space it administers. This should be done for whole groups of objects with particular properties rather than on a per object basis for scalability reasons. In our example, the local trader might "announce" that it administers route table entries with the *routerId=router-A* property. This property does not map to a route attribute of the relevant interface but this is not necessary in CORBA: any properties may be dynamically associated to interfaces. These interactions are shown as "export" and "federate" in Figure 4-9.

When the manager object wants to access those route objects, it will have to contact its local trader and ask for interfaces with particular properties. The relevant operation could be expressed as: *import (ifType=routeEntry, properties={router=router-A, nextHopAddr=X}).* The trader should scan the relevant properties and based on the fact that *router=router-A* has been specified it should federate the request to the trader in the router. This federation may also involve other intermediate traders but only the two traders mentioned are shown in Figure 4-9. The trader in the router will return the relevant interface references to the initial trader which will subsequently return them to the manager. If dynamically changing attributes are involved in the assertion, the trader may have to "search" through the relevant objects and retrieve the values of those attributes; this is not the case in the example presented here.

Having retrieved the interface references, the manager will perform the operations to the relevant route entries according to its management policy e.g. retrieve the route destination and quality attributes, delete the routes, etc. These operations will have to be performed concurrently in order to minimise the overall latency. Given the fact that CORBA operations are performed synchronously, a multi-threaded execution environment is of paramount importance. The minimum management traffic incurred will be: 4 RPC packets for trading (2 to/from the domain trader and at least another 2 between the latter and the trader in the router); and 2*N RPC packets for retrieving the entry attributes i.e. 2*(N+2) packets in total. The equivalent amount of traffic using OSI-SM was N+2 packets as described in section 2.2.4 of Chapter 2. On the other hand, the amount of data in those CORBA packets may be less than in the equivalent OSI-SM packets. We will investigate the exact amount of traffic incurred by CORBA in section 4.5.3.

We will emphasise again the fact that this solution is only hypothetical at present, assuming traders that support rich query facilities, and more important, federation. As such, it can not be provided with the current state of ODP / CORBA technology. Supporting trader federation in an optimal fashion is an interesting research issue that needs to be addressed. The analysis in this section has been presented in [Pav97a] and [Pav97d].

4.4.3.2 A TMN-Oriented Approach Using Discovery Through Naming and Containment

While in the previous approach we adopted a pure ODP view, we will now look at another, TMN-oriented approach that can be thought as being at the other end of the spectrum.

This approach assumes the same hierarchical naming scheme as in OSI-SM / TMN systems, based on the GDMO prescribed name bindings in "agent" domains and on the OSI Directory global name schema specified in [X750]. For example, the name of the root object in a CORBA managed object cluster that constitutes a TMN OS could be:
{ c=GB, o=UCL, ou=CS, cn=NM-OS, systemId=NM-OS }
This is now an instance of the CORBA CosNaming::Name IDL type as specified by the OMG Name Service [COSS]. Both OMG and OSI-SM names are ordered lists of *type=value* components, so there can be a direct mapping between the two. The key difference is that the OMG name space is generally an acyclic graph instead of a hierarchical tree. Since we are adopting the TMN hierarchical naming principles, the OMG *management* name space becomes a hierarchical tree.

The first four components of the above example name denote *naming contexts,* i.e. they are not associated to CORBA objects. The fifth component, i.e. systemId=NM-OS, is a name bound to the compound context defined by the previous four. These contexts and the relevant name will be registered in the CORBA naming service [COSS]. A client or manager object will be able to resolve the object's name to an interface reference through the naming server. In addition, the client will be also able to discover all the management applications running in the UCL CS domain by performing a *list* operation on the naming context *{ c=GB, o=UCL, ou=CS }*. It will subsequently be able to obtain the subordinate object names of those contexts, e.g. *systemId=NM-OS* for the *cn=NM-OS* context, and resolve them to interface references. This architecture provides discovery functionality similar to that of the OSI Directory in OSI-SM / TMN environments [X750]. In addition, it can be supported by the current state of CORBA through the use of naming services [COSS]. The latter may be federated in order to cope with large name spaces and different administrative domains. Federated name services can be easily

provided in comparison to federated trading because of the hierarchical structure of the name space.

Having presented the system discovery aspects, we also need to address discovery facilities within an MIT cluster. Every managed object is aware of its name, which will be passed to it at by its "factory" at creation time. In addition, every managed object is aware of its superior and subordinate objects. Those object references form now a "virtual" MIT, since the relevant managed objects may be physically distributed across different network nodes. The superior reference is passed to an object at creation time. The subordinate references are passed to an object by the subordinate objects themselves, which update their superior at creation and deletion and maintain the "referential integrity" of the MIT. The Code 4-2 caption shows a potential IDL specification of the i_ManagedObject interface that supports this functionality.

```
interface i_ManagedObject
{
    CosNaming::Name   getName ();
    CosNaming::NameComponent
                      getRelativeName ();
    i_ManagedObject   resolve (in CosNaming::Name name)
                            raises (NotFound);

    i_ManagedObject   getSuperior ();
    ManagedObjectList getSubordinates ();
    oneway void       getSubordinatesAsync
                        (in i_ManagerObject manager, in long invId);

    void              addSubordinate (in i_ManagedObject subord)
                            raises (InvalidObject);
    void              removeSubordinate (in i_ManagedObject subord)
                            raises (InvalidObject);

    void              destroy ()
                            raises (NotDeletable,
                                    DeleteContainedObjects);
};
```

Code 4-2 The i_ManagedObject IDL Interface

Every managed object provides access to the references of its superior and immediately subordinate objects in the MIT. It can *resolve* a subordinate name to a reference by using recursively the *getSubordinates* and *getRelativeName* methods.

Manager objects may discover the exact structure of the MIT, starting from the root object and using those discovery facilities. This approach is in fact similar to the OSI-SM / TMN one apart from scoping and filtering. It should be noted that every object acts as a name server for objects in its subtree. The key advantage of the approach is that managed objects other than the MIT root do not have to register with the name service; this results in fewer interactions across the network and more timely operation. If the name service was used instead, subordinate names would have to be retrieved from the name server through a "list" operation and subsequently mapped to

interfaces references through a "resolve" operation. In this architecture, the CORBA name service is only used for getting access to the root MIT object.

The problem with this approach is that the advantages of the ODP trading or OSI-SM discovery through scoping and filtering are lost. In the example presented before, the client object would have to retrieve the whole routing table, identify the desired routing entries and subsequently perform operations on them. In this respect, the approach is more akin to the Internet SNMP rather than to OSI-SM.

We could have added scoping at least to the i_ManagedObject interface, since it can be easily supported by traversing the "contains" relationship through the getSubordinates method. The problem though is one of "culture": scoping is a CMIS-related facility while the i_ManagedObject interface is totally unrelated to CMIS as an access method. Adding scoping, filtering and the full OSI-SM access functionality over CORBA is the next step. We will address the relevant issues in section 4.4.4.

It should be finally noted that the *getSubordinatesAsync* method is an engineering optimisation. Object references are fairly big as it will be discussed in the performance analysis section. As a result, the size of the GIOP response packet will be very big for the getSubordinates method. The getSubordinatesAsync method instructs the managed object to start "pushing" the references one by one in the opposite direction. The symmetric method for receiving the result is supported by the *i_ManagerObject* interface.

4.4.3.3 Adding Object Lifecycle and Event Dissemination Aspects

In the previous two sections we concentrated on the discovery and access aspects, adopting first a pure ODP and then a minimal OSI-SM approach over CORBA. Two other important aspects of a management framework are object lifecycle, i.e. managed object creation and deletion, and event dissemination. We will address those here, taking a realistic approach which uses existing CORBA facilities. As such, the relevant design completes the second approach presented above.

In every "agent" domain, there will exist a *factory finder* object, bound to the domain naming context e.g. cn=NM-OS. A client will be able to obtain its name from the name server through a "list" operation and resolve it subsequently to an interface reference. A further optimisation can be achieved by agreeing in advance on the relative name of the factory finders e.g. *factoryFinderId=NULL*, since there will always exist a single instance in a domain. The factory finder will provide access to specific factories for a particular type of interface, e.g. i_uxObj, as advocated in the CORBA lifecycle services [COSS]. Specific factory interfaces will exist for

every GDMO class that has *create* properties. A factory interface will take parameters according to the GDMO class specification, which may include the name of the object to create, its superior's name, a "reference" object and initial values for attributes. A factory interface bears similarities to the CMIS *m-create* primitive but initial attribute values can be strongly typed. An example factory interface for the i_uxObj is shown in the Code 4-3 caption.

```
interface i_uxObjFactory
{
    i_uxObj  create_with_name(in CosNaming::Name uxObjName,
                              in string wiseSayingValue)
                      raises (invalidName, duplicateName);

    i_uxObj  create_with_automatic_name
                              (in string wiseSayingValue,
                               out CosNaming::Name uxObjName);
};
```

Code 4-3 The i_uxObjFactory IDL Interface

Managed object deletion is supported through the *destroy* method of the i_ManagedObject interface. Derived implementation classes will have to redefine the relevant method behaviour according to the GDMO name binding properties i.e. deny deletion, instruct the caller to delete contained objects first or delete the whole subtree. As it was mentioned in section 4.3.3.2, operations in which the object destroys itself are not well supported in current CORBA implementations: a standard exception is generated and the client object does not know if the server object cleaned up and died properly or it left a mess behind. This behaviour will be hopefully corrected in future implementations.

The final important point for a complete CORBA-based architecture is event dissemination. This can be based on the existing OMG *event services* [COSS]. In every "agent" domain, there will exist a *channel finder* object, in a similar fashion to the factory finder one. This will provide access to event channels that serve specific event types. Managed objects that generate notifications will register with the corresponding event channels as "event suppliers". Manager objects will get access first to the channel finder through the naming service and then to the particular event-specific channels, registering as "event consumers". Generated notifications will be sent to the corresponding channels and will be subsequently passed to the manager objects using the push or pull model. The fact that event channels correspond to particular event types can support strongly typed event dissemination. Event type specific push and pull interfaces will be produced for every GDMO notification and will be supported by the relevant channels.

In summary, this event reporting approach adopts the CORBA event services. This means that the "agent" does not support EFD [X734] and log managed objects. The drawback is the functionality of event filtering and event logging are lost. Event logging could be provided by

modifying event channels to get access to the MIT, check log objects and create log records through the relevant factory interface. In this case, the log filter should only contain assertions on the event type. While this approach is feasible, it requires the modification of the CORBA event channels [COSS].

Let's consider again the routing table example we presented before. The manager object is interested to know of any added or deleted routes that point to a particular next hop address. Assuming that *routeEntryCreation* and *routeEntryDeletion* events are defined, the manager will have to register with the relevant event channels. It will subsequently receive all the route entry creation and deletion events and will have to select locally those of interest, assuming that the next hop address property is contained in the notification. While this approach generates additional management traffic in comparison to the OSI-SM event model, it provides a workable solution based on the current OMG facilities.

4.4.3.4 Summary and the Proposed Architecture

In the previous sections we looked at the issues of using CORBA instead of OSI-SM as the base technology for TMN systems. We presented first a pure ODP-based approach for object discovery through trading. This cannot be supported with the current state of technology because OMG trading services are not finalised, they do not support rich query facilities and federation issues have not been addressed. We then presented a OSI-SM-based discovery approach, which uses hierarchical naming and containment relationships while the OMG naming service is used to locate the root object, the factory finder and the event channel finder in every "agent" domain. In this approach OMG event services support event dissemination. The relevant architecture is depicted in Figure 4-10, showing the various interactions as described in the previous sections.

The key advantage of using CORBA is that the managed objects that constitute a logical "agent" cluster may be distributed across different "capsules", i.e. operating system processes, which may in turn be distributed across different network nodes. The event channel finder and event channels will be located in the same capsule. The managed object factories will be located in the capsules where the relevant interfaces will be created. The factory finder will be statically configured to know the references of the relevant factories in that domain.

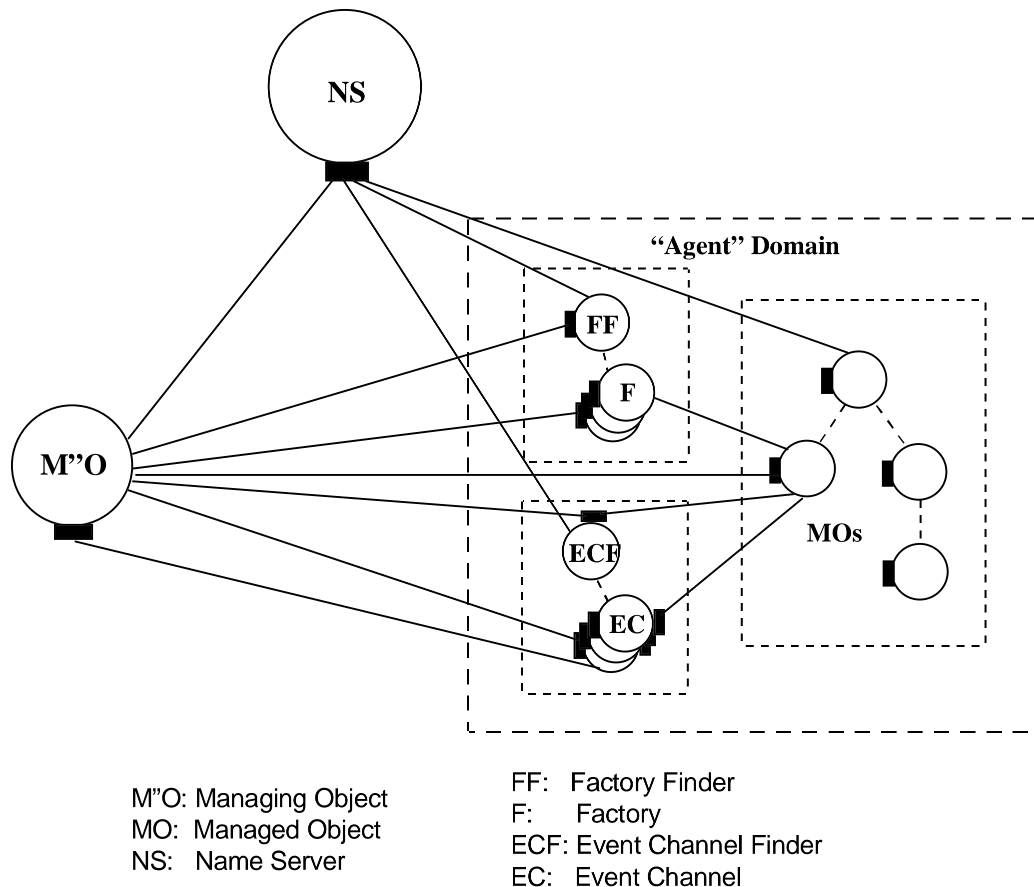


Figure 4-10 A Basic Architecture for OSI-SM to CORBA Mapping

The approach presented is feasible with the current state of CORBA technology. The most difficult aspect to realise is the functionality of the i_ManagedObject interface. This is in fact a subset of the MO class of the OSIMIS GMS, which was presented in section 3.6 of Chapter 3. The disadvantages in comparison to OSI-SM are the following:

- there is no support for multiple attribute access;
- there is no support for multiple object access through one management operation;
- object discovery facilities do not support scoping and filtering;
- events are disseminated based on the event type i.e. there is no support for filtering;
- there is no support for logging; and
- non-DII operations on objects can only be synchronous, which requires support for multi-threaded operation.

In the next section we will examine how to address those disadvantages, reproducing the full OSI-SM functionality over CORBA.

4.4.4 A Complete Mapping of the OSI-SM Model to CORBA

Since the JIDM work on the mapping between GDMO and CORBA IDL was first published in 1994, a number of researchers started investigating the issues behind a CORBA-based management architecture. The work presented in the previous section is a part of the research work done by the author on this topic. The rest of the research work, which completes the proposed framework, will be presented in this section. We will examine first the related work by other researchers, presenting it in chronological order.

4.4.4.1 Related Research Work

[Fuen94] discusses the application of the TINA ODP-based architecture to management services. It presents the view that management applications should be modelled by OSI-SM-like agents, which are computational objects with IDL interfaces in the TINA management architecture. Managed objects do not have their own computational interface but are specified as information objects in Quasi-GDMO and realised as engineering objects within the agent.

The GDMO to CORBA IDL mapping presented in [JIDM95] addresses the static translation aspects. The architecture of a management environment based on the resulting CORBA specifications is another issue. [Mazum96] presents the first research work on such an architecture as a proposal to the JIDM group. The first version of this work appeared in 1995 and tries to re-use as much as possible the existing OMG services. The key element of this approach is that it establishes a “shared management knowledge” repository in CORBA, which recaptures aspects of the GDMO specification lost in the translation e.g. the MATCHES FOR properties of attributes. The OMG property service is used to support filtering in a complicated fashion while the TINA-specified notification service is used in conjunction with EFDs for event dissemination. The proposed approach is very complex and has not been adopted by the JIDM group.

[Geni96] is a PhD thesis titled “Towards a Distributed Architecture for Systems Management”. It addresses the mapping of OSI-SM to ODP principles by using the JIDM translation approach, which it applies to ANSA instead of CORBA IDL. The presented concepts have been validated through the implementation of bidirectional gateways between ANSA and OSI-SM [Geni95]. The problem with the approach is that it takes a pure ANSA / ODP view, in which a lot of the OSI-SM functionality is lost, namely multiple attribute access, multiple operations to objects through scoping and filtering and fine-grain control of notifications through filtering and event logging.

The author started working in this area in 1995 as a result of his involvement in the ACTS VITAL project. The latter tries to validate the emerging TINA framework for service control and network management [TINA]. The author's initial approach was to model OSI-SM agents as computational entities with CMIS-like IDL interfaces, based on the initial ideas in [Fuen94] but taking those to completion as presented here. The first version of the relevant architecture and specification was released within the VITAL project in April 1996 [Pav96e]. As a result, the author was pointed to a TINA-C group working in a very similar direction and proposing a similar architecture [Garc96].

[Garc96] realises that the full functionality of OSI-SM needs to be preserved over distributed object frameworks because of its importance for telecommunications management environments. It proposes that managed objects are grouped together in "agent" clusters and named using TMN-based hierarchical naming principles. In addition, it proposes those to be administered by a Management Broker (MB), which is a computational entity similar to an OSI-SM agent. The latter offers a CMIS-like interface which supports multiple attribute access and multiple object access through scoping and filtering. Event reporting and logging are supported through EFD and log managed objects. This approach was only a paper exercise that never went into considerations behind the potential realisation of such a framework. As such, it was never taken any further within TINA-C.

The author architected a very similar approach which could be realised based on the OSIMIS environment and its reusable software components. A first implementation of a generic gateway between CORBA and OSI-SM was produced in the summer of 1996 by T. Tin of UCL and was used in the VITAL project [Pav97b][Huel97]. At the same time, the author was discussing this approach with the TMN research group of the Hewlett-Packard Research Laboratory at Bristol (HPLB). One of the HPLB researchers supervised a MSc project at the University of Twente, which verified the feasibility of the framework with another implementation [Hars96].

Finally, [Schad96] proposes a model for distributed applications management which tries to extend CORBA with multiple object access and fine-grain event filtering facilities. Multiple operations are supported by "multicast objects" while the fine grain event model is based on a combination of EFDs with the CORBA event services.

Having presented all the relevant research work in this area, the obvious question is which approach did the JIDM group finally adopt for the CORBA to OSI-SM gateways. The answer is none of the above. [Hier96] is a relatively new proposal which has been adopted. While different from the other official JIDM proposal [Mazum96], it combines elements of the other approaches.

Managed objects are organised in “agent” domains and are named hierarchically. Event dissemination is handled through a specialisation of the OMG event service, using event channels in both manager and agent domains but with no filtering and logging at present. Multiple attribute and multiple object access is supported through the JIDM i_ManagedObject interface which is CMIS-like. This means that every managed object acts as an agent or management broker for its subtree. This approach is different to the [Pav97b] and [Garc96], we will discuss the benefit of a separate management broker later.

4.4.4.2 Adding Multiple Attribute Access and Filtering

We will now examine if and how the elusive full OSI-SM functionality can be added to the framework presented in section 4.4.3.4.

We will start first with multiple attribute access. With the current GDMO to IDL mapping, every attribute is mapped to one or more access methods. As a consequence, manager objects have to access attributes on a one-by-one basis, which creates unnecessary management traffic. Accessing multiple attributes is an important management requirement. In addition, many applications use the CMIS “get all attributes” facility, which should also be supported. The obvious place to put this functionality is the i_ManagedObject interface, but how can this be supported?

The key problem is knowing what the attributes of a managed object instance are. The i_ManagedObject part of a MO instance could interrogate the CORBA interface repository for the attributes of every derived interface and access them locally, through the DII. Unfortunately, this approach will not work. The problem is that as a result of the GDMO to IDL translation, the notion of attributes is lost. This means that the CORBA interface repository cannot be used. An alternative approach would be to provide “shared management knowledge” about the information of a GDMO-derived IDL interface. For example, this information is stored in a *discovery* managed object in OSI-SM / TMN environments [X750]; [Mazum96] proposes such an approach.

A third and most efficient approach would be similar to that of OSIMIS and most TMN platforms: every derived implementation class should pass the names of its attributes to the i_ManagedObject part of an instance at creation time. The only problem with this approach is that this code will need to be hand-written, which is both tedious and error prone. In TMN platforms, this code is automatically produced by the GDMO compiler. A way around this

problem would be the existence of special "JIDM-aware" IDL compilers which could produce this code automatically.

In summary, it is possible to support multiple attribute access. The example method signatures for getting multiple attributes are shown in the Code 4-4 caption. A similar *setAttributes* method could also be provided. It should be finally noted that the resulting methods are weakly-typed because the IDL *any* type is used for attribute values.

```
typedef string AttributeId_t;
typedef sequence<AttributeId_t> AttributeIdList_t;

struct Attribute {
    AttributeId_t attrId;
    any           attrVal;
};

enum GetListError_t {
    noError,
    noSuchAttribute
};

struct GetAttribute_t {
    GetListError_t error;
    AttributeId_t  attrId;
    any            attrVal;      // "empty" in errors
};
typedef sequence<GetAttribute_t> GetAttributeList_t;


interface i_ManagedObject
{
    // . . .

    void     getAttributes    (in  AttributeIdList_t  attrIdList,
                               out GetAttributeList_t attrList);

    void     getAllAttributes (out AttributeList_t attrList);


    boolean  evaluateFilter (in Filter_t filter);

};
```

Code 4-4 Multiple Attribute Access and Filtering

The next aspect to consider is filtering, which is a much more difficult proposition. [Mazum96] proposes to use the OMG *property* service, together with "shared management knowledge" which provides access to the GDMO MATCHES FOR properties of attributes. The solution is very complex, not clearly presented and has not been validated through implementation. [Hier96] specifies filtering as part of the CMIS-like access methods of the i_ManagedObject interface but does not discuss at all how it is going to be provided. It should be noted that supporting filtering in CORBA to OSI-SM gateways is easy since the filter will be evaluated in the target OSI-SM agent; this is not the case in native CORBA environments.

Let's revisit first how filtering is supported in OSI-SM environments and examine how the same functionality could be provided in CORBA. Filter assertions on a particular attribute are evaluated by the attribute itself. The ASN.1 compiler produces a *compare* method which tests the data type for equality, while ordering and substring testing have to be added by hand. In multi-valued attributes, equality concerns the contained elements. Produced classes derive from a generic class, i.e. *Attr* in OSIMIS, which can evaluate a filter assertion by using the compare method and methods to "walk through" a multi-valued attribute. This was described in section 3.4 of Chapter 3. The generic MO class can evaluate filter expressions since it keeps "handles" to attributes of derived classes, which evaluate specific assertions in the filter.

The problem with CORBA is that attributes are not "first class citizens" of the framework. Defining an attribute in an IDL interface results in nothing more than access methods being produced, without any special support for the relevant data type. As such, there is no automatic support for equality comparison and subsequently for the evaluation of filter assertions. One solution to this problem would be for OMG to consider providing such support through a special extension to IDL. Types preceded by some special keyword, e.g. `attribute`, could be treated specially, deriving from a generic attribute class and supporting equality assertions. This requires though the modification of both the IDL and the relevant programming language mappings.

[Mazum96] mentions that the comparison methods required for filtering could be either provided by hand, which is obviously not desirable, or produced by modified IDL compilers which understand comment lines with special significance. The Alcatel trader implementation [Trate96] uses string property values in order to be able to support equality testing. It also supports ordering assertions on attributes but only for string, integer and real types. These are "cast back" to the precise type according to the *type code* that characterises instances of the CORBA *any* type.

In summary, it is not easy to provide filtering in native CORBA environments. In general, the mapping of IDL types to object classes is not rich enough, lacking support for comparison, pretty-printing and other generic functionality. As such, it is problematic to deal with instances of the *any* type. This is an area that needs special attention by the OMG if CORBA is to become the ubiquitous object infrastructure. We will assume for the time being that filtering is possible and that the i_ManagedObject interface supports the *evaluateFilter* method as it was shown in the Code 4-4 caption This is in accordance to the OSI-SM management information model [X720] which suggests that filtering should be supported by a managed object. The Filter_t is the IDL type that maps to the CMISFilter ASN.1 type of [X711] according to the JIDM rules.

Given the support for filtering and the fact that containment relationships can be navigated through the *getSuperior* and *getSubordinates* methods, multiple access to managed objects and sophisticated discovery facilities can be provided. The relevant functionality is similar to that provided by OSI-SM agents and the question is where it should be placed. Both [Mazum96] and [Hier96], i.e. the two JIDM proposals, suggest that it should be part of the i_ManagedObject interface. This essentially means that every managed object behaves as an agent for its subtree. In contrast, the author [Pav97b][Pav97d] and [Garc96] propose to separate this functionality from the managed objects, so that different sophisticated access styles can be provided. A detailed analysis of the second approach is presented in the next section.

4.4.4.3 Fine-grain Event Dissemination and Multiple Object Access Through the Management Broker

The i_ManagedObject interface supports the following functionality as presented in the previous sections:

- name resolution for contained objects;
- access to the "containedIn" and "contains" relationships;
- filter evaluation;
- get and set access to multiple attributes through a single access method; and
- object deletion.

This functionality is exactly as prescribed by the OSI-SM MIM [X720] and supported in the OSIMIS by the GMS MO class. This approach essentially "externalises" a managed object which now becomes "first class citizen" of the proposed distributed management framework.

ODP purists may argue that some of the above managed object functionality is "OSI-SM oriented" and not in the spirit of ODP, in particular the hierarchical naming, containment relationships and filtering. Hierarchical naming schemes are used in many contexts and not only in OSI-SM. For example, the postal and network addresses and even the OMG naming architecture use hierarchical schemes. Managed objects may exhibit many other relationships in addition to containment. These can be realised either through "pointer" attributes, in a similar fashion to containment, or through separate relationship objects [X725] that map onto the OMG relationship service [COSS]. Finally, filtering provides a powerful mechanism to express sophisticated assertions on attributes and properties. The ODP / OMG trading service might also support filtering in the future.

Given the support for filtering, fine-grain event reporting and logging can be provided by migrating the relevant OSI-SM models over CORBA. In every "agent" domain, there will exist a Event Processing (EP) object. Managed objects will get access it through local means, e.g. the factories may pass its reference to MOs at creation time. MOs will subsequently "push" their notifications to it. The EP object will create the "potential event report / log record" through the relevant object factory, evaluate the filters of EFDs and logs and may instruct the latter to send the event or log it as a log record accordingly. This is exactly the behaviour prescribed in [X734] and [X735]. Note that the existence of the EP object is totally transparent to manager objects that are interested to receive event reports.

Manager objects will request the forwarding of events by creating EFDs and setting the *destination* attribute to contain either their name or object reference. This implies that the syntax of the *destination* and *backupDestinationList* EFD attributes [X734] will have to be slightly modified. Destinations are currently specified as OSI-SM distinguished names which are equivalent to CORBA names, but CORBA object references should be added. This approach retains the full power and expressiveness of the OSI-SM event reporting and logging. A pictorial view of this approach is presented in Figure 4-11.

The last aspect of the OSI-SM / TMN framework that needs to be provided is support for multiple object discovery and access facilities based on scoping and filtering. Such access facilities are certainly "OSI-SM / TMN specific" and should be provided in an incremental fashion, without being an integral aspect of the rest of the framework. A key reason for considering those separately is they are do not represent the only way of providing this type of functionality. For example, in the CMIS/P access model containment relationships are navigated first through scoping with filtering applied at the end of the selection process. [Pav97e] proposes an enhanced model in which any relationships could be navigated, with filtering possibly applied at various stages of the object selection process. Other mechanisms may be invented in the future that suit best the needs of particular management environments.

This is the why the author proposes to separate the CMIS-based access aspects from the rest of the management framework. As such, CMIS-based access should *not* be part of the i_ManagedObject interface but should be supported by a separate *Management Broker* (MB) object. Given the fact that CMIS is the current access mechanism in TMN environments, a MB should always exist in an "agent" domain with its name bound to the domain naming context e.g. *{ c=GB, o=UCL, ou=CS, cn=NM-OS, brokerId=CMIS }*. Managed objects could be accessed either directly or through the MB. The advantage of MB-based access is object discovery and

multiple object access through scoping and filtering. The disadvantage is that the relevant access paradigm is weakly-typed: attribute and action values are of the IDL *any* type. The architecture of the proposed framework is depicted in Figure 4-11, including the event dissemination through EFDs. This updates the architecture that was presented in Figure 4-10.

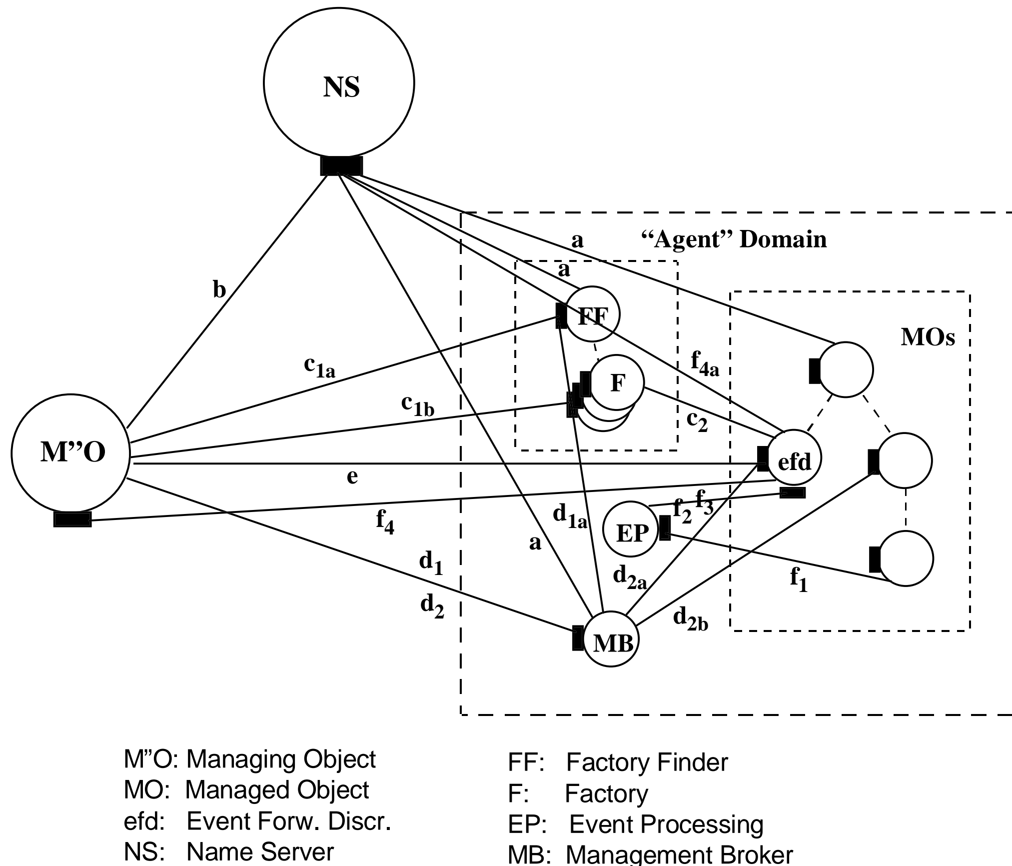


Figure 4-11 A Complete Architecture for OSI-SM to CORBA Mapping

When an "agent" domain is instantiated, the root MIT MO, the factory finder and the management broker register themselves with the name service (interactions *a* in the figure). Manager objects need to know in advance the domain name, e.g. *{ c=GB, o=UCL, ou=CS, cn=NM-OS }*. They may invoke a list operation on the name service and discover the names and subsequently the references of the MIT root, FF and MB objects (interaction *b*). A MO may be created either in a strongly-typed fashion through the relevant factory (interactions c_{1a} and c_{1b}) or in a generic, weakly-typed fashion through the MB (interactions d_1 and d_{1a}). The manager may subsequently access the object either directly (interaction e) or through the MB. In the latter case it will probably access more than one MOs e.g. to suspend the operation of all its EFDs (interactions d_2, d_{2a} and d_{2b}). A MO emits a notification by "pushing" it to the event processing object (interaction f_1). The latter will create first a "potential event report", retrieve an EFD's

filter (interaction f_2) and evaluate it. The potential event report is not shown since it is manipulated locally by the EP i.e. can be thought as encapsulated by it. If the filter evaluates to true, it will instruct the EFD to send the event report (interaction f_3). The EFD may need to resolve the name of the manager to an interface reference through the name service (interaction f_{4a}) and "push" the event to the manager (interaction f_4).

In order to make the use of the framework more concrete, we will consider the same example we considered in section 4.4.3.1 and also in section 2.2.4 of Chapter 2. The manager object in this case will know the naming context of the router e.g. *{ c=GB, o=UCL, ou=CS, cn=router-A }*. By concatenating the relative name *brokerId=CMIS*, it will be able to obtain a reference to the broker object. It could then perform a *multipleObjectGetAsync* operation (shown later in the Code 4-6 caption) to the object with name *{ subsystemId=nw, protEntityId=clnp, tableId=route }* and request the first level subordinates that satisfy the filter *(nextHopAddr=X)*. The amount of traffic incurred in terms of packets will be exactly the same as in OSI-SM.

An alternative approach would be to perform the same operations without the management broker. The relative name of the root MIT object will be *managedElement=router-A*, so its name can be resolved to an interface reference through the name server. A subsequent resolve operation to the managedElement object will return an interface reference to the table object and then a *getSubordinates* operation will retrieve the subordinate references. The route entries will then be accessed one by one in order to identify the ones with the particular destination address. The problem in this case is that the traffic incurred will be much more than the previous approach, mainly because of the lack of filtering. In addition, a lot of object references need to be communicated and these are big as the measurements show in section 4.5.3.

Finally, the manager may request event reporting by creating an EFD object with the same filter as in OSI-SM. The EFD destination attribute may be initialised with the object reference of the manager and not only with its name. The advantage of the object reference is that it avoids name resolution through the name server, which is otherwise necessary. The EFD may be created in a weakly-typed fashion through the management broker, or in a strongly-typed fashion through the relevant factory. The reference of the factory will be obtained through the factory finder.

Returning to the proposed architecture, different management brokers may be specified and implemented in the future, providing varying styles of object discovery and multiple object access. With the advent of highly portable languages like Java [Sun96] and associated mobile code paradigms, managers may be able to install dynamically new management brokers in an agent domain to suit their needs. OMG is currently attempting to specify a framework for mobile

code known as the Mobile Agent System Interoperability Facility (MASIF). The current management broker provides CMIS-based access since CMIS/P is currently the basis for the TMN Q_3 and X interfaces.

We describe below the broker's CMIS-like interface, examining the issues of mapping CMIS/P to CORBA IDL. The simplest form of management operations the MB provides are the CMIS m-get, m-set, m-action, m-delete and m-create, applied to a single managed object. These operations are also supported by the managed objects through the specific IDL interface that results from the GDMO to IDL translation, e.g. i_uxObj and i_uxObjFactory, and also through the generic i_ManagedObject interface. The reasons for providing the same functionality through the CMIS management broker are twofold:

a) we can exploit the CORBA *oneway* operations and provide an asynchronous interface - the interfaces resulting from GDMO to IDL translation according to the JIDM rules are only synchronous; and

b) the management broker may play the role of a CORBA-based management agent with the managed objects being plain engineering objects i.e. *without* IDL interfaces; we will examine this style of organisation in the next section.

The Code 4-5 caption shows the specification of the equivalent CMIS m-get and m-action methods in IDL, *get*, *getAsync* and *action*, *actionAsync* respectively. In the case of the asynchronous operation, the invoking manager object passes its reference for the result or error to be sent back. It also passes an invokeId argument, which can be used to distinguish between results / errors in the case of many outstanding invocations; this is equivalent to the CMISE/ROSE invokeId. The *objectClass* parameter may be used to force allomorphic behaviour. This is possible though only in environments like b) above, since allomorphism is meaningless for MOs with native CORBA IDL interfaces.

```
interface i_CMISBroker
{
    CosNaming::Name getName ();  // the broker's name

    // . . .

    void     get (
             in  CosNaming::Name    objectName,
             in  string             objectClass,    // for allomorphism
             in  AttributeIdList_t  attrIdList,     // empty -> "all"
             out GetAttributeList_t attrList
             )
             raises (GET_ERRORS);

    onway void getAsync (
             in  i_CMISManager      managerRef,
             in  long               invokeId,
             in  CosNaming::Name    objectName,
             in  string             objectClass,    // for allomorphism
             in  AttributeIdList_t  attrIdList,     // empty -> "all"
             );

    void     action (
             in  CosNaming::Name    objectName,
             in  string             objectClass,    // for allomorphism
             in  string             actionType,
             in  any                actionInfo,
             out any                actionReply
             )
             raises (ACTION_ERRORS);

    onway void actionAsync (
             in  i_CMISManager      managerRef,
             in  long               invokeId,
             in  CosNaming::Name    objectName,
             in  string             objectClass,    // for allomorphism
             in  string             actionType,
             in  any                actionInfo
             );

};
```

Code 4-5 Single Object CMIS-like Access in IDL

The next type of management operations are object discovery based on scoping and filtering and multiple object access for the m-get, m-set, m-action and m-delete operations. The synchronous version of the object discovery and the synchronous and asynchronous versions of the multiple object get and action operations are shown in the Code 4-6 caption. The *objectSelection* operation returns both the managed object's name and interface reference. In the case of environments like b) described above, only the name only is returned while the reference will be always "nil".

```
// Scope_t, Filter_t and Sync_t map exactly to the
// X.711 Scope, CMISFilter and CMISSync ASN.1 types

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

typedef struct ObjectNameList_t {
    CosNaming::Name  name;
    Object           objectRef;
};

interface i_CMISBroker {

// . . .

    void     objectSelection (
                    in  CosNaming::Name   baseObjectName,
                    in  ObjectSelection_t objectSelection,
                    out ObjectNameList_t  objectNameList
             ) raises (OBJECT_SELECTION_ERRORS);

    void     multipleObjectGet (
                    in  CosNaming::Name   baseObjectName,
                    in  ObjectSelection_t objectSelection,
                    in  Sync_t            sync,
                    in  AttributeIdList_t attrIdList, // empty -> "all"
                    out GetResultList_t   resultList
             ) raises (MULTIPLE_OBJ_OPER_ERRORS);

    void     multipleObjectGetAsync (
                    in  i_CMISManager     managerRef,
                    in  long              invokeId,
                    in  CosNaming::Name   baseObjectName,
                    in  ObjectSelection_t objectSelection,
                    in  Sync_t            sync,
                    in  AttributeIdList_t attrIdList, // empty -> "all"
             );

    void     multipleObjectAction (
                    in  CosNaming::Name   baseObjectName,
                    in  ObjectSelection_t objectSelection,
                    in  Sync_t            sync,
                    in  string            actionType,
                    in  any               actionInfo,
                    out ActionResultList_t resultList
             ) raises (MULTIPLE_OBJ_OPER_ERRORS);

    void     multipleObjectActionAsync (
                    in  i_CMISManager     managerRef,
                    in  long              invokeId,
                    in  CosNaming::Name   baseObjectName,
                    in  ObjectSelection_t objectSelection,
                    in  Sync_t            sync,
                    in  string            actionType,
                    in  any               actionInfo,
             );

};
```

Code 4-6 Multiple Object CMIS-like Access in IDL

Finally, the Code 4-7 caption shows a part of the interface that should be supported by CMIS manager objects in order to receive asynchronous results and event reports. The latter may be both confirmed and non-confirmed. Note also that the event reports contain the name of the EFD that triggered them, so that manager applications may "demultiplex" them; this reflects an extension to CMIS/P that was proposed in Chapter 3.

```
interface i_CMISManager {

// . . .

    oneway void endOfLinkedReplies (
                    in long invokeId
            );

    oneway void getResult (
                    in long invokeId,
                    in boolean linked,
                    in GetResult_t result
            );

    oneway void actionResult (
                    in long invokeId,
                    in boolean linked,
                    in ActionResult_t result
            );

    oneway void eventReportUnconfirmed (
                    in  string           objectClass,
                    in  CosNaming::Name  objectName,
                    in  CosNaming::Name  efdName,
                    in  UTCTime_t        eventTime,
                    in  string           eventType,
                    in  any              eventInfo   // optional
            );

    void     eventReport (
                    in  string           objectClass,
                    in  CosNaming::Name  objectName,
                    in  CosNaming::Name  efdName,
                    in  UTCTime_t        eventTime,
                    in  string           eventType,
                    in  any              eventInfo,  // optional
                    out any              eventReply  // optional
            ) raises (EVENT_REPORT_ERRORS);
};
```

Code 4-7 The Generic CMIS-like Manager Interface

In summary, mapping CMIS to CORBA IDL interfaces is relatively straightforward. The author thought of the approach and produced the relevant specification in early 1996 [Pav96e]. [Garc96] and [Hier96] have also proposed similar conceptual approaches which mainly differ in the IDL mappings for CMIS. The author's approach tries to stay as close to CMIS as possible but introduces a number of simplifications, based on the OSIMIS design and implementation experience.

4.4.4.4 Design, Implementation and OSI-SM to CORBA Migration Aspects

In the previous sections we discussed the issues behind the mapping of the OSI-SM / TMN model onto emerging distributed object frameworks, using OMG CORBA as the target framework. We presented a complete architecture which is in the spirit of ODP / CORBA but which retains the power and expressiveness of OSI-SM, necessary for TMN environments. The proposed architecture uses only the OMG naming service from the common object services [COSS] and is implementable with the current state of CORBA, apart from automated support for filtering. The advantage of the CORBA-based architecture is that it will support the distribution of parts of TMN applications, which currently operate on a single network node. It might also be more performant that Q_3 protocol stacks; we will examine this assertion in the next section. Finally, a single technology will be used in TMN environments instead of the combination of OSI-SM [X700] and the OSI Directory [X500].

Given the target CORBA-based framework that was depicted in Figure 4-11, we will examine how this can be implemented. We will consider in particular a phased approach which reuses initially parts of the OSI-SM based infrastructure described in Chapter 3. Since OSI-SM is currently the base TMN technology, it is important to devise a phased transition strategy that will ease compatibility and interoperability with existing TMN systems and will reuse as much as possible existing TMN platform components.

The first step for migrating towards the target framework is to support only agent discovery and CMIS interactions through CORBA, without individual IDL interfaces for managed objects. This essentially means that the management broker will act as an agent that provides access to managed objects which are implemented by existing TMN platform infrastructure i.e. GDMO/ASN.1 compilers and relevant APIs. The MB may be used in conjunction to the existing Q_3 agent object within an agent application e.g. the CMISAgent object in the OSIMIS GMS. In this case, the TMN application in agent role will have two interfaces: the existing Q_3 interface and the CORBA version of the "Q_3" interface as specified by the i_CMISBroker and i_CMISManager IDL interfaces.

This minimal approach is depicted in Figure 4-13 and has no impact at all at the implementation of managed objects which are based on TMN platform technology e.g. OSIMIS. Existing OSI-based manager applications will continue to function while new CORBA-based management applications may be developed. CORBA manager objects get access to the MB interface reference through the CORBA naming services [COSS] while OSI manager objects get access to

the presentation address of the OSI agent through the OSI directory [X750]. It should be noted that two different notations have been used in this figure to depict interactions, one for CORBA using object interfaces and one for OSI-SM using arrows. In addition, the CORBA interface notation belongs to computational viewpoint while the TMN agent application is depicted from an engineering viewpoint. This "mismatch" is necessary in order to avoid a more complicated drawing.

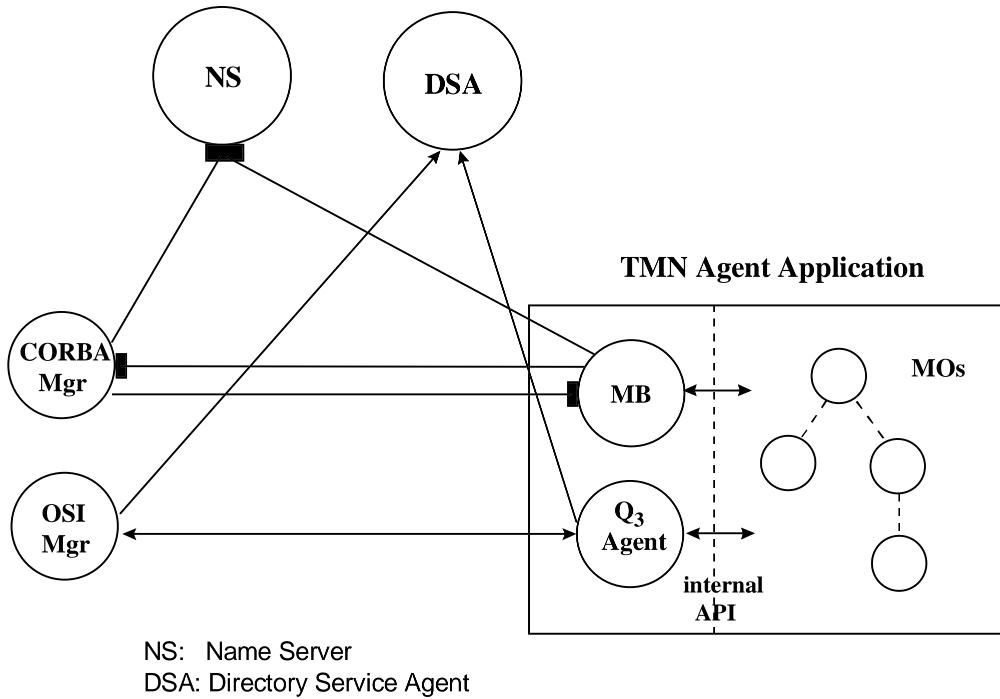


Figure 4-12 Dual Q_3 and CORBA Agent

This architecture exploits the fact that the object models are the same in the two frameworks, and provides a "dual-agent" access paradigm. Dual agents support different access methods for semantically equivalent managed objects [Newk94]. In this framework, the managed object GDMO/ASN.1 specifications are translated to IDL but the resulting IDL interface specifications are not instantiated: they simply "document" the management broker interface which provides access to those objects in a dynamic, weakly-typed fashion. The attribute, action and event ASN.1 types are used across the management broker interface through the equivalent IDL types.

A variation of this approach is the gateway approach, in which the management broker becomes a separate application which accesses one or more management agents in the "back-end" through their Q_3 interfaces. The gateway approach is useful to provide adaptation for TMN applications that are already deployed, in which case it is not possible to add to them the management broker

in a tightly-coupled fashion. The disadvantage of that approach is reduced performance, which is in general the case with adaptation mechanisms.

T. Tin of UCL together with the author implemented a gateway version of the MB during the summer of 1996. This was subsequently used in the first trial of the VITAL project in October 1996. This was the first CORBA to CMIS/P gateway that provided the full OSI-SM functionality. Since then, TMN platform vendors have announced similar products based on the emerging JIDM proposal for the interaction translation [Hier96]. The author subsequently designed and implemented the tightly-coupled dual agent approach depicted in Figure 4-12. [Hauw97] also reports a very similar dual-agent approach. CORBA-based agent applications were used in the second trial of the VITAL project [Pav97b] and the first trial of the REFORM project [Sartz97].

Implementing the gateway was fairly straightforward. The only difficulty encountered was the bi-directional translation between ASN.1 and IDL data types. This can be automated if one has access to a flexible ASN.1 compiler, which could be customised to produce the equivalent IDL types as well as the conversion routines in both directions. Since OSIMIS uses the ISODE pepsy compiler which cannot be easily customised, these conversions had to be hand-coded. Implementing the tightly-coupled dual agent version was also straightforward given the well-defined OSIMIS APIs for accessing managed objects within an agent application. Based on this approach, existing OSIMIS agent applications can be made to work over CORBA by changing a few lines of code in the main program and re-linking them with the management broker library.

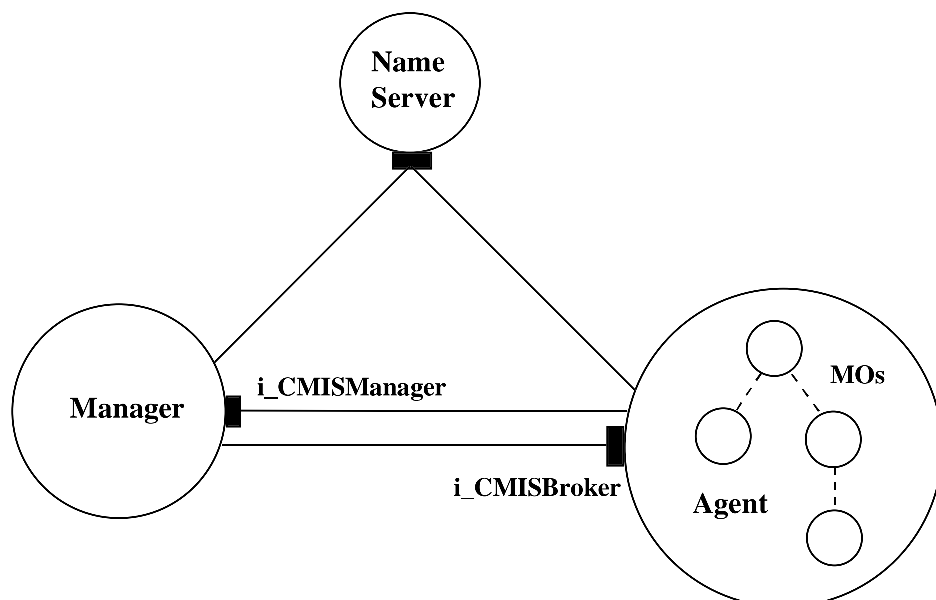


Figure 4-13 The Management Broker as an Agent Computational Object

Another way to look at this paradigm in which managed objects are accessed indirectly through a management broker is depicted in Figure 4-13. In this case the managed objects can be thought as encapsulated by the MB. The latter is a computational object in ODP terms which provides the agent aspects of the CMIS service through the i_CMISBroker interface. The manager supports the manager aspects through the i_CMISManager interface. Managed objects are specified in GDMO in the information viewpoint but do not manifest themselves in the computational viewpoint. They become directly engineering objects administered by the MB, in a similar fashion to OSI-SM agents. This is another view for the mapping the OSI-SM model to ODP.

The complete CORBA-based framework requires also that individual managed objects become computational constructs with IDL interfaces. The advantage in this case is strongly-typed access, superior distribution aspects and better performance as will be presented in section 4.5.2. Having produced the relevant specifications using GDMO to IDL translation, the key issue is how to support the functionality of the i_ManagedObject interface. The difficult aspects regarding this realisation are the “get all” attributes and the attribute filtering; we explain how these can be implemented below.

Classes for derived interfaces should pass to their i_ManagedObject parent class the names and types of their attributes when an object instance is created. The i_ManagedObject part will subsequently access the attributes of derived parts as if they belonged to separate objects, through the dynamic invocation interface. This scheme supports the multiple attribute get and set methods.

The only way to deal with filtering is to provide the IDL compare and “multi-valued” attribute traversal routines by hand, in a hash table indexed by the relevant type. The i_ManagedObject part will retrieve the attributes involved in the filter and will invoke the relevant compare and traverse methods, getting access to them through the attribute type which is known from its “repository”. While this is feasible, it is far from desirable since it requires hand-written routines for every attribute type. On the other hand this is the only approach to support filtering until OMG considers the inclusion of relevant features in IDL and their concrete support through the programming language mappings.

Despite the fact that the author has not validated the above concepts through implementation, they are largely based on the design concepts of the managed objects in the OSIMIS GMS, which *have* been validated through implementation.

4.5 Performance Analysis and Evaluation

In this section we attempt a performance analysis and evaluation of the CORBA-based framework. We measure performance aspects of both native CORBA managed objects and of the management broker introduced in the previous section. Since one of the main motivations for migrating to CORBA is a potential performance advantage over OSI-SM, we attempt to verify if this is the case. Another potential advantage is the superior distribution framework and we attempt to evaluate its cost in terms of computing resources.

The measurements address the same aspects as for the OSI-SM framework, namely program size, response times and amount of communicated information. A commercial CORBA platform was used for the development and measurements. As such, it was not possible to measure aspects "beneath" the APIs available to the developer. This means is was impossible to scrutinise the end-to-end performance and attribute it to various parts of the infrastructure as we did for OSI-SM.

The measurements of the CORBA framework also aim to put into perspective the equivalent OSI-SM measurements. While we assessed the OSI-SM framework as performant given the low response times and the relatively modest memory requirements, those figures were absolute and were judged in relatively subjective terms. In this section, we compare and contrast those figures to the equivalent CORBA ones, so we will be able to re-evaluate the performance of OSI-SM in the context of this comparison.

The environment and methodology for the experiments were similar to those used for the OSI-SM experiments. These have been described in section 3.8.1 of Chapter 3. The *Orbix 2.2* implementation of CORBA was used, a leading CORBA product marketed by IONA technologies. Since there exist various programming language mappings for CORBA, the C++ mapping was used in the experiments. In addition, the standard IIOP protocol suite was used for interoperable communication. The management broker was based on Orbix 2.2 and the OSIMIS-4.0 Generic Managed System (GMS). The experiments were performed on the Sun Sparc 5 / Sparc 20 pair of workstations running Solaris. It was not possible to repeat them on the 486 PCs since Orbix does not work on the Linux version of UNIX.

4.5.1 Program Size

Every network node that can host CORBA programs needs to run the local part of the ORB. This is equivalent to the ODP concept of nucleus and in Orbix is implemented by a daemon process, the *orbixd*. The latter is around 2700 Kb at run time. There is no equivalent daemon in the OSI-SM environment since the upper layer stack is directly linked with the applications and the agent plays to some extent the role of the ORB.

We will examine first the size of a CORBA server process containing managed objects with native IDL interfaces. A CORBA server is equivalent to the ODP concept of capsule. We will consider the *i_uxObj* interface that was depicted the in Code 4-1 caption and is fully specified in Appendix C. The only difference is that there is no i_ManagedObject interface i.e. the inheritance hierarchy is *i_uxObj -> i_top -> CORBA::Object*. This is exactly equivalent the uxObj GDMO class, so that we can draw relevant comparisons. The factory for this interface is *i_uxObjFactory*, as it was specified in the Code 4-3 caption.

The size of the server process that contains one instance of the *i_uxObjFactory* and the *i_uxObj* interfaces, with the attribute values for the latter as specified in the Code 3-12 caption of Chapter 3, is 2640 Kb at run time. The equivalent size of an OSI-SM agent with the same functionality was 1340 Kb but the OSI agent supports also scoping, filtering, event reporting and logging. In addition, the orbixd is needed on that node, which increases the overall memory requirements. The size of the smallest possible server is 2520 Kb, measured for a server with an i_echo interface that supports a simple echo operation.

The amount of code linked in for the *i_uxObjFactory* and the *i_uxObj* interfaces is about 80 Kb at run time while the equivalent amount for the uxObj class was around 20 Kb. In the latter case there is no need for a specific factory class since the agent plays the role of a generic factory.

The data overhead of a *i_uxObj* instance with the attribute values as above is exactly 896 bytes. The equivalent overhead of a GDMO *uxObj* instance was 420 bytes. In the latter case though the object is part of the MIT and can evaluate filters; this functionality incurs additional memory overhead due to the required instance variables.

The size of the client program that invokes an echo action on the i_uxObj interface is 2540 Kb at run time. The equivalent size of the OSI-SM manager program was 1060 Kb.

Finally, these experiments were performed without the CORBA name server: the server program printed out the i_uxObj reference to a file and the client program read it from there. When the

name server is used, the size of both the client and server programs are slightly bigger since they would need to link in the name server's interface. The size of the name server that handles a naming graph of modest size is about 2800 Kb at run time.

All the above measurements concern native CORBA managed objects. If the management broker approach is followed, the size of the *i_CMISBroker* server object is comparable to the i_uxObj server object. The overhead of managed object instances in this case is the same as in the OSI-SM approach, since they are held internally as OSIMIS managed objects.

Description	Size (Kb)
ORB daemon	2700
Server object size (without name server access)	2640
Client object size (without name server access)	2540
Overhead of the i_uxObj / i_uxObjFactory interfaces	80
Overhead of the uxObj instance as in Code 3-12	0.896
Small Name Server size	2800

Table 4-2 Summary of the CORBA Memory Overheads

Table 4-2 presents the summary of the program size overheads for the native CORBA based approach. In summary, the overheads are at least double of those of OSI-SM which were presented in Table 3-7 of Chapter 3. The key problem is the size of server object: the smallest possible server is around 2.5 Mb at run-time. The initial idea behind ODP was to be able to distribute objects as much as possible so that they can benefit from the various transparencies independently, without being clustered together. This type of distribution requires many servers to host those objects but the size of a server seems to be prohibitive for fine grain distribution. While this program size is acceptable for the ORB daemon, it is too big for server objects.

On the other hand, the memory overhead of an object instance within a server process is relatively reasonable, though twice the size of that of OSIMIS. The key reason for the bigger overhead compared to OSI-SM is the size of the IIOP Inter-Operable Reference (IOR), which is at least 200 bytes. [Whit97] accepts this as an issue that affects CORBA scalability and proposes compositional models of *entities* and *warehouses* so that the number of references are reduced. The management broker approach is in fact such a compositional model.

4.5.2 Response Times

We will now examine the response times for the *echo* operation of the *i_uxObj* interface. Before the operation takes place, the client object needs to establish an interface reference to the object in server role. This is typically done by resolving a name to an interface reference through the naming server, as already discussed. After that, the client performs directly the operation without the need to explicitly establish a connection to the server. Since IIOP uses the Internet TCP/IP, a TCP connection will need to be established by the infrastructure but this is totally transparent to the object. Since a connection is implicitly established, an increased response time is experienced the first time a client accesses a server object. We can thus calculate the response time for establishing an underlying connection but we cannot calculate the response time for releasing it.

Performing an echo operation for the "hello" string results in 5 msecs response time. The equivalent operation on the uxObj instance using the Q_3 interface was 11 msecs. Note that the figure for the latter presented in the Table 3-9 of Chapter 3 was 12 msecs but the operation was performed asynchronously: the synchronous operation is about 8% faster as explained in Chapter 3 or 11 msecs. This is a significant performance advantage compared to OSIMIS i.e. 45% of its access time. The explanation for this lies mostly in the supporting protocol stacks and is according to expectations. The CMIS m-action request and response messages contain other parameters in addition to the action type and echoed string, e.g. the object class, name, time etc., and their encoding and decoding causes additional overhead.

Increasing gradually the size of the echoed string results in an almost linear increase of the response time. The additional overhead is around 0.3 msecs per 100 bytes or another 0.06% of the overall response time. The equivalent figure for OSI-SM is 0.21 msecs per 100 bytes. The fact the OSI-SM figure is smaller than the CORBA one is unexpected. A detailed comparison between the CORBA encoding rules and the BER is required to investigate this further.

The first time the operation is performed it takes 17 msecs because the underlying TCP connection is established. This implies that connection establishment takes roughly 17-5 = 12 msecs. The equivalent time for establishing a CMIS association in OSIMIS is 39 msecs. This is again very favourable for CORBA i.e. roughly 30% of the OSI-SM association establishment time. It can be easily explained because of the various negotiations that take place in the OSI upper layers, involving a number of packet exchanges as described in Chapter 3.

We will now examine the same echo operation performed indirectly to the managed object through a synchronous operation to the *i_CMISBroker* interface. The *action* method signature has been depicted in the Code 4-5 caption and is equivalent to a "synchronous version" of the

CMIS m-action primitive: the object name and class are passed in addition to the action type and action information. This operation takes 8 msecs in comparison to 11 msecs through the Q_3 interface i.e. 72% of the Q_3 access time. This operation is more expensive than accessing directly the i_uxObj interface for two reasons: first, additional parameters are passed across a similar fashion to CMIS/P; and second, a mapping is necessary between the IDL C++ data types and the equivalent OSIMIS ASN.1 C++ types.

Performing the same operation asynchronously takes 9.2 msecs in comparison to 12 msecs through the Q_3 interface, which amounts to 77% of the Q_3 access time. The signatures for the "one way" *actionAsync* and *actionResult* operations have been depicted in the Code 4-5 and Code 4-7 captions respectively. The additional overhead in this case is again because of two reasons: first, additional parameters are passed across in both the request and the response primitives; and second, asynchronous operations are always slightly slower than synchronous ones as it was discovered in Chapter 3. It should be finally stated that performing an operation asynchronously is exactly equivalent to the corresponding CMIS operation: the same parameters are passed across in the request and response primitives.

[Schmi97] has also addressed CORBA performance. The relevant experiments in that case were done over ATM using the most powerful Sun Sparc stations. Figures as low as 2 msecs for a parameterless operation are reported. [Park96] has also compared a CORBA implementation with OSIMIS and concludes that CORBA is around 40% faster, which is not far from the figures presented above.

In summary, CORBA performance is better than OSI-SM when measuring the Orbix and OSIMIS implementations respectively. This is particularly true when managed objects with direct IDL interfaces are used. When a CMIS-like approach through the management broker is used, the performance difference is smaller. While these performance differences are not dramatic, there is ongoing work towards real-time ORBs that will improve CORBA performance further. On the other hand, the proximity of the Orbix and OSIMIS performance figures verifies the good performance of OSIMIS and OSI-SM in general in this comparative context.

4.5.3 Packet Sizes

We finally consider the sizes of management packets, which include the payload over the TCP as explained in section 3.8.1 of Chapter 3.

Establishing a connection between objects in different network nodes does not involve any additional procedures to TCP connection establishment. This means that only two TCP packets are exchanged with zero payload. In contrast, OSI-SM requires 4 additional packets which contain 310 bytes of data as it was shown in Table 3-10. The same is true for connection release: OSI-SM requires 3 additional packets which contain 64 bytes of data as it was shown in Table 3-11. In summary, connection establishment and release in the CORBA IIOP relies on the underlying TCP procedures. As such, it generates much less traffic that OSI-SM.

Performing an echo operation with an empty string on the i_uxObj interface results in a request packet of 101 bytes and a response packet of 29 bytes. Performing the same operation through the synchronous management broker interface results in request and response packets of 177 and 37 bytes respectively. Finally, performing the same operation through the asynchronous management broker interface results in request and response packets of 409 and 97 bytes respectively. This last operation is exactly equivalent to the CMIS/P one for which the relevant sizes are 72 and 88 bytes respectively as it was shown in Table 3-12.

The interesting result here is that CORBA seems to generates a relatively large amount of traffic. The payload for the simplest operation is 101/29 bytes and these packet sizes increase significantly with additional argument and result parameters. This increase becomes much bigger when IORs are passed across, as experienced in the asynchronous operation in which the request packet size “shot up” to 409 bytes. This is expected because of the size of IORs. It appears though that the CORBA encoding rules are at least as verbose as the BER [X209].

These measurements also verify the fact the size OSI-SM operation packets is relatively modest, despite the number of parameters required by CMIS/P.

4.6 Summary

4.6.1 Overview of this Chapter

In this chapter we presented first an analysis of ODP as the theoretical framework for distributed systems. In this analysis, we discussed OSI-SM and TMN from an ODP perspective and presented briefly two different approaches for mapping OSI-SM to ODP. We then presented ANSA, the OSF DCE and OMG CORBA as three generations of ODP-influenced technologies. An analysis showed that both ANSA and the OSF DCE failed to satisfy essential TMN requirements i.e. object-orientation and dynamic operation without pre-compiled knowledge. On the other hand, OMG CORBA seems to be the first ODP-influenced technology that has come of age, satisfying most of the distributed framework requirements identified in Chapter 3. A drawback is that its IDL abstract language does not support polymorphic distributed operations.

Given the emergence of CORBA as the ubiquitous object technology for open distributed systems, we examined in detail how it can be used as the base technology for the TMN. We started from an ODP-influenced approach in which trading is used instead of hierarchical naming for object discovery and demonstrated the problems with this approach in management environments with large sizes of distributed objects. We subsequently presented a minimal approach which retains the TMN hierarchical naming and containment relationships but does not support scoping, filtering, multiple object access and fine-grain event reporting and logging. A key aspect is that only few objects in each “agent” domain need to export their names to the name server. This avoids scalability problems in TMN environments where network elements and operations systems may contain 10’s of thousands of objects.

We then added multiple attribute access and filtering to the managed objects and explained how CMIS-like multiple object access can be supported through the management broker. This was done in an incremental fashion, without mixing CMIS-like access aspects with the managed object interfaces. We finally exploited the filtering capability of managed objects in order to add EFD-based fine-grain event reporting. The proposed architecture retains the advantages of OSI-SM but with the following drawbacks: support for filtering and knowledge about the attributes of a particular object need to be hand-coded i.e. they cannot be automatically supported by IDL compilers. OMG may re-consider its attribute model in the future and add expressiveness similar to GDMO; this will solve these problems. The management broker approach was validated through implementation and was used in project trials.

Finally, a brief performance analysis and evaluation of the CORBA-based framework was attempted with the main target to compare the Orbix implementation of CORBA to the OSIMIS implementation of OSI-SM. The results have shown CORBA to exhibit faster access times than OSI-SM, in particular when accessing managed objects directly. Access through the management broker is slightly slower but still faster than OSI-SM. The penalty for this increased performance seems to be the memory requirements: the size of a server process is too big and this discourages fine-grain partitioning of objects into servers for distribution purposes. In addition, the memory overhead of an object instance is at least twice that of OSI-SM. Finally, the traffic incurred seems to be at best similar to OSI-SM and much bigger when object references are communicated.

In summary, CORBA seems a very promising technology which can form the basis of future TMN systems. Its value compared to OSI-SM is not so much the slightly better performance but the fact that it may become the ubiquitous technology for future heterogeneous distributed systems. Its use in both management and service control environments will result in economies of scale and allow for easier integration of new managed resources.

We should finally answer the question of what is the architectural impact to the TMN if CORBA is adopted. The answer is that there is no impact at all. The TMN architecture and methodologies will remain exactly as presented in Chapter 2. Interface specifications will be produced in GDMO, following possibly guidelines which will guarantee that generic translation to IDL is possible. In addition, the Q_3 interface profiles will be modified, including CORBA protocols as valid choices for TMN interfaces. The use of CMIS/P-GDMO or GIOP-IDL will become an engineering issue for implementing the same specifications. It should be added though that mappings of GIOP over lower level protocols other than TCP/IP are necessary in order to meet the needs of telecommunications environments. A GIOP mapping to SS7 is currently being addressed by OMG. Finally, an additional benefit of using CORBA is that TMN OS components could be distributed across different network nodes.

In summary, CORBA can be seen as an object-oriented version of ROSE which can support any transaction-based application layer protocols. It was shown in this chapter how CMISE can be supported through the CORBA-based management broker. The addition of stream interfaces and quality of service parameters [Hand95][Kits95] will make CORBA the ubiquitous infrastructure that will be able to support any application layer protocols, involving stream transfers as well as operational transactions.

4.6.2 Research Contribution

The first part of this chapter introduced the ODP framework and associated technologies in the context of telecommunications management, concentrating mostly on OMG CORBA. Despite the fact that the relevant sections served mainly as state-of-the-art presentations, they were presented from the point of view of suitability for telecommunications management and, as such, they constitute to some extent research contributions in their own right. In this section we state clearly the research contributions in this chapter.

- The suggestion of two different approaches for describing OSI-SM / TMN in ODP terms in section 4.2.5. The second approach, in which an OSI-SM/TMN agent becomes a computational object, was elaborated further through the Management Broker approach.
- The analysis of the properties of ANSA and OSF DCE as candidate technologies for telecommunications management in sections 4.3.1 and 4.3.2 respectively. The conclusion that they failed to satisfy important requirements.
- The analysis of the properties of OMG CORBA as a candidate technology for telecommunications management in section 4.3.3. The identification of deficiencies such as true polymorphic operation, lack of built-in support for asynchronous operations and not expressive enough attribute and event models. On the other hand, the finding that CORBA satisfies most of the desired properties of distributed object frameworks and can be used as base technology for the TMN.
- The discussion of the JIDM mapping rules between GDMO and CORBA IDL [JIDM95] in section 4.4.2 and the identification of problematic aspects regarding the mapping of attributes and conditional packages.
- The analysis of how the OSI-SM / TMN model could be mapped to CORBA in a pure ODP fashion which uses trading and avoids hierarchical naming schemes in section 4.4.3.1. This approach needs federated trading which is very difficult to provide in non-hierarchical service spaces. In addition, the OSI-SM / TMN access aspects are lost and this results in increased management traffic.
- A similar analysis which retains the OSI-SM / TMN hierarchical naming and containment relationships for managed objects in section 4.4.3.2 and the proposed "minimal" architecture in section 4.4.3.4. This can be easily provided with the current state of CORBA technology and is unrelated to CMIS as an access method.

- The analysis of how the previous minimal architecture can be enhanced with multiple object access through scoping and filtering and fine-grain event dissemination in section 4.4.4. The presentation of the Management Broker approach which adds facilities other than filtering in an orthogonal fashion to managed objects. The complete specification, implementation and validation of the management broker approach.
- The presentation of an evolutionary approach towards a CORBA-based environment in section 4.4.4.4. In this, management brokers can be implemented in various stages: first as gateways, then as dual agents, subsequently as plain CORBA-based agents and finally as brokers for CORBA-based MOs with native IDL interfaces.
- Finally, the performance analysis and evaluation of the CORBA-based framework in section 4.5 and its comparison to the OSI-SM one. The main findings were that CORBA servers and objects are relatively big, packet sizes are not small but the performance is very good, especially when accessing MOs directly, through their native IDL interfaces.

It should be finally mentioned that the research contributions in this chapter are the author's alone. The starting point for this research work has been the work of the JIDM group on the comparison of the OSI-SM and OMG CORBA object models and the guidelines for translating OSI-SM GDMO definitions to CORBA IDL. The author would like to thank T. Rutt of Lucent Technologies, the editor of the comparison of the object models document [Rutt94]; S. Mazumdar, also of Lucent Technologies, the editor of the GDMO/ASN.1 to IDL translation document [JIDM95]; and T. Tin of UCL for implementing the gateway version of the management broker.