Using Distributed Object Technologies in Telecommunications Network Management

George Pavlou

***Abstract*—The Telecommunications Management Network (TMN) has been developed as the framework to support administrations in managing telecommunications networks. It suggests the use of OSI Systems Management (OSI-SM) as the technology for management information exchanges. Distributed object technologies, such as the Common Object Request Broker architecture (CORBA), address the use of software application program interfaces (API's) in addition to interoperable protocols. Their use in TMN has been the subject of intensive research in recent years, with most approaches focusing on interoperability aspects with OSI-SM. In this paper we examine the issues behind using distributed object technologies in TMN via a native fashion, with network elements supporting distributed objects directly, e.g., a "CORBA to the switch" approach. The proposed solution tries to maintain the full OSI-SM expressive power in a way that other solutions have not attempted before. Performance and scalability issues are considered, while the approach has been validated through implementation.**

***Index Terms*—CORBA, distributed objects, ODP, OSI-SM, TMN.**

I. Introduction

THE TELECOMMUNICATIONS Management Network (TMN) [1] has been developed as the framework to support administrations in managing telecommunications networks. It suggests the use of OSI Systems Management (OSI-SM) [2] as the technology for management information exchanges. The latter follows an object-oriented approach in terms of information specification, but leaves aspects related to the software structure of relevant applications unspecified. Distributed object technologies address the use of software application program interfaces (API's) in addition to interoperable protocols. Their ease of use, generality, and ubiquity implies that they might also be used in telecommunications network management.

Since the early inception of Open Distributed Processing (ODP) [3], 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 the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) [4]. Given the fact that the TMN is a large scale distributed system, it is valid to consider the use of CORBA as its base technology, replacing OSI-SM [2] and the OSI Directory [5]. The relationship between OSI-SM and CORBA has attracted considerable attention from the research community in recent years. Most approaches have focused on interoperability aspects with OSI-SM, assuming the latter will be used in elements and the TMN element and network management layers. In this paper we propose a solution that maintains the full expressive power of OSI-SM and provides a smooth migration path toward a CORBA-based TMN. Although CORBA is used as the representative distributed object technology, the proposed framework is general enough to be applicable to other similar technologies.

Key motivations for using CORBA in TMN environments are the following. OSI-SM was conceived as an object-oriented management technology in the absence of a general purpose distributed object-oriented framework. CORBA provides exactly such a framework, with a superior distribution paradigm in which every object could be potentially distributed. Its performance could also be better than OSI-SM due to a more lightweight protocol stack—this assertion is assessed in this paper. Finally, CORBA exhibits multiple standard mappings to O-O programming languages while most OSI-SM platforms support mainly C++ API's. On the other hand, OSI-SM exhibits a more expressive object model, superior object access which allows multiple operations in a single request through scoping and filtering, and a scalable event dissemination model based on fine-grain event selection criteria. As such, the mapping of OSI-SM to CORBA presents a difficult technical challenge.

The solution proposed in this paper relies on the initial Joint Inter-Domain Management (JIDM) work for the static mapping [6] of the OSI-SM Guidelines for the Definition of Managed Objects (GDMO) [7] to the CORBA Interface Definition Language (IDL) [4]. The issues behind this mapping and its implications are discussed in Section II. An initial mapping of the OSI-SM model to CORBA is presented in Section III. This is enhanced to a complete mapping in Section IV which retains the full OSI-SM/TMN expressive power. Section V discusses design, implementation, and OSI-SM to CORBA migration aspects. Section VI investigates performance and scalability aspects. Finally, Section VII presents a summary.

II. Mapping OSI Systems Management to Open Distributed Processing

A. Mapping OSI-SM to ODP

Since the early days of ODP, there have been various attempts to describe OSI-SM in ODP terms. The first attempt is

Manuscript received April 1, 1999; revised December 14, 1999. This work was undertaken by the ACTS REFORM and FlowThru projects, supported in part by the Commission of the European Union.

The author is with the Center for Communication Systems Research, School of Electronic Engineering and Information Technology, University of Surrey, Guildford, Surrey GU2 5XH U.K. (e-mail: G.Pavlou@eim.surrey.ac.uk).

Publisher Item Identifier S 0733-8716(00)02763-3.

described in [9], considering managed and managing[1] objects as ODP objects. This implies that functionality of OSI-SM agents needs to be supported via ODP mechanisms, i.e., through special server objects similar to the CORBA common object services [10].

A similar approach has been more recently standardized in the Open Distributed Management Architecture (ODMA) [8], which is the ISO/ITU-T approach to describe OSI-SM in ODP terms. ODMA tries to provide a generic management framework that can be mapped to either ODP-based object technologies or to OSI-SM communication protocols. OSI managed and managing objects map onto ODP objects and interfaces, while 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 platform, managing and managed objects communicate directly with each other. When the underlying platform is OSI-SM-based, the OSI agent becomes an "operation dispatcher" in the engineering viewpoint that performs operations to managed objects. It also becomes a "notification dispatcher" that forwards notifications to managing systems.

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, 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 API's. The intention is to allow for the specification of management systems from an information and computational perspective in an engineering-neutral fashion.

We could characterize 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, [8] recognizes the fact that multiple object access through scoping and filtering and event dissemination through filtering and event forwarding discriminators may need to be exposed in the computational viewpoint. This leaves open the possibility for other potential mappings between OSI-SM and ODP. We present such an approach and explain in detail the relevant issues during the rest of this paper.

B. Mapping OSI-SM GDMO Objects to IDL Interfaces

Mappings between GDMO and CORBA IDL have been addressed by JIDM. Although 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 native CORBA-based management systems, preserving the large body of existing GDMO specifications for a number of different network technologies. 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 [11]. The *specification translation* aspects followed [6], including the generic mapping of GDMO to CORBA IDL.

While IDL interfaces have attributes similar 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., $\langle attr \rangle$*_get*, $\langle attr \rangle$*_set*, $\langle attr \rangle$*_setToDefault*, $\langle attr \rangle$*_add*, and $\langle attr \rangle$*_remove*.

GDMO actions can be naturally mapped to IDL methods with input argument the action information and output argument the action result. Action parameters, which signify action-specific errors, are mapped to IDL exceptions. GDMO notifications can be mapped to 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. It should be noted that the OSI-SM notification model corresponds to the CORBA "push" model, although a "pull" model could be emulated through logging [13] and subsequent access of log records.

A key 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. Their presence, however, will become an implementation issue. CORBA supports a standard *not_implemented* exception which will be raised whenever a method of a nonsupported 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 [14].

Given the rules for GDMO to IDL 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 it is investigated in this paper. 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 [14]. 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 Fig. 1.

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 immediately subordinate objects in the containment hierarchy.

[1]The term "managing object" is used rather loosely, since OSI-SM defines only the term "manager" [2], with objects in the manager system not defined. ODMA though refers to managing objects [8].

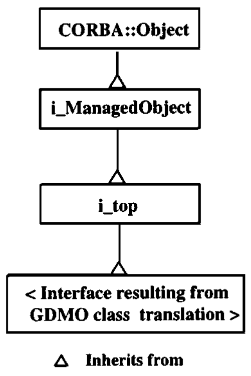

Fig. 1. Inheritance hierarchy resulting from GDMO to IDL mapping.

III. An Initial Mapping of the OSI-SM Model to Corba

A. Object Discovery Through Hierarchical Naming and Containment Relationships

This approach assumes the same hierarchical naming scheme as in OSI-SM/TMN systems, based on the GDMO name bindings in "agent" domains and on the OSI Directory global name schema specified in X.750 [15]. For example, the name of the root object in a CORBA managed object cluster that constitutes a TMN Operations System (OS) could be

$$\{c = GB,\, o = UniS,\, ou = CCSR,\\ cn = NM - OS,\, systemId = NM - OS\}.$$

This is now an instance of the CORBA CosNaming::Name IDL type as specified by the OMG Name Service [10]. Both OMG and OSI-SM names are ordered lists of *type=value* components, so there can be a direct mapping between the two. The OMG {*id*, *value*} tuple can map to an X.500/OSI-SM Relative Distinguished Name (RDN). The key difference is that the OMG name space is generally a 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*. The fifth component, i.e., systemId=NM-OS, is a name bound to the compound context structure defined by the previous four. These contexts and the relevant name will be registered with the CORBA naming service [10]. 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 UniS CCSR domain by performing a *list* operation on the naming context $\{c = GB, o = UoS, ou = CCSR\}$. This architecture provides discovery functionality similar to that of the OSI Directory in OSI-SM/TMN environments [15], but it is supported through the use of naming services [10].

Having presented the system discovery aspects, we also need to address discovery facilities within a Management Information Tree (MIT) cluster. Every managed object is aware of its name, which will be passed to it 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. Code 1 (Fig. 2) shows a potential IDL specification of the i_ManagedObject interface that supports this functionality. The *addSubordinate* method is used by factory objects when the superior is in a different ODP "capsule," i.e., distributed program implemented as an operating system process, containing CORBA objects. Similarly, the *removeSubordinate* method is used by a superior object during object deletion when the subordinate is in a different capsule. Finally, the *destroy* method implements CMIS M-Delete semantics.

Every managed object provides access to the references of its superior (*getSuperior*) and first level subordinate objects in the MIT (*getSubordinates*). Through the resolve method, it can resolve a subordinate name to a reference; this is possible by retrieving the relative names at every MIT level and comparing them to the expected relative name for that level.

Manager objects may discover the exact structure of the MIT, starting from the root object and using these facilities. This approach is in fact similar to the OSI-SM/TMN one, apart from scoping and filtering. 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 faster operation.

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, however, is one of "culture": scoping is a facility related to the OSI-SM Common Management Information Service (CMIS) [16] while the i_ManagedObject interface is totally unrelated to CMIS as an access method. Adding scoping, filtering, and the full OSI-SM access functionality to the proposed framework will be considered in Section IV.

B. Object Lifecycle and Event Dissemination Aspects

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 optimization 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 as advocated by the CORBA lifecycle services [10]. 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. Managed object deletion is supported through the *destroy* method of the i_ManagedObject interface.

```
typedef sequence<i_ManagedObject> ManagedObjectList;

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

    i_ManagedObject         getSuperior ();
    ManagedObjectList       getSubordinates ();

    void                    addSubordinate (in i_ManagedObject subord)
                                    raises (InvalidObject);
    void                    removeSubordinate (in i_ManagedObject subord)
                                    raises (InvalidObject);
    void                    destroy ()
                                    raises (NotDeletable, DeleteContainedObjects);
};
```

Fig. 2. Code 1: The i_ManagedObject IDL Interface.

The final important point for a complete CORBA-based architecture is event dissemination. This can be based on the existing OMG *event services* [10]. 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 specific 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.

C. The Proposed Architecture

The relevant architecture is depicted in Fig. 3, showing the various interactions as described before. 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 disadvantages of this approach 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.

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

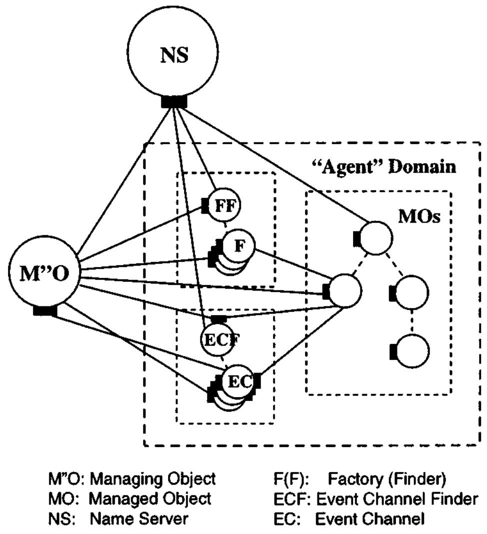

Fig. 3. A basic architecture for OSI-SM to CORBA mapping.

IV. A Complete Mapping of the OSI-SM Model to CORBA

A. Related Research Work

We will examine first the most important related research work, presenting it in chronological order. Reference [17] discusses the application of the TINA ODP-based architecture to management services. It presents the view that management applications should be modeled by OSI-SM-like agents, which are computational objects with IDL interfaces. Managed objects do not have their own computational interface, but are specified as information objects and mapped to engineering objects within the agent.

The GDMO-to-CORBA IDL mapping presented in [6] addresses the static translation aspects. The architecture of a management environment based on the resulting CORBA specifications is another issue. Reference [18] 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 reuse as much as possible the existing OMG services.

The author's initial approach (circa 1996) was to model OSI-SM agents as computational entities with CMIS-like IDL

```
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);

};
```

Fig. 4. Code 2. Multiple Attribute Access and Filtering.

interfaces, based on the initial ideas in [17], but taking those to completion. The first version of the relevant architecture and specification was produced in the ACTS VITAL project which validated the TINA framework.

Reference [21] 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 Event Forwarding Discriminator (EFD) and log managed objects.

The author architected a very similar approach which could be realized based on the OSIMIS environment [22] and its reusable software components. A first implementation of a generic gateway between CORBA and OSI-SM was produced in 1996. A second implementation followed in 1997, with native CORBA-based management agents [20] as described here.

Finally, [19] influenced mostly the official JIDM approach. While different from the other initial JIDM proposal [18], it combines elements of the other approaches. Managed objects are organized in "agent" domains and are named hierarchically. Event dissemination is handled through a specialization of the OMG event service, using event channels in both manager and agent domains. Multiple attribute and multiple object access is supported through the JIDM i_ManagedObject interface which is CMIS-like. This approach is different from both [20] and [21], and requires many more interactions across the network, in contrast to the approach presented here.

B. Multiple Attribute Access and Filtering

Accessing multiple attributes with one operation 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.

The key problem is knowing what the attributes of a managed object instance are. The i_ManagedObject part of an 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, which 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 [15]; [18] proposes such an approach.

A third and most efficient approach would be similar to that of 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 handwritten, which is both tedious and error prone, while in TMN platforms it is automatically produced by GDMO compilers. A way around this problem would be the existence of special "JIDM-aware" IDL compilers which could produce this code automatically. The method signature for getting multiple attributes is shown in Code 2 (Fig. 4). A similar *setAttributes* method could also be provided.

The next aspect to consider is filtering, which is a much more difficult proposition. Reference [18] proposes to use the OMG *property* service, together with "shared management knowledge" which provides access to the GDMO MATCHES FOR properties of attributes, a solution which is very complex. Reference [19] 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. Supporting filtering in CORBA to OSI-SM gateways is easy since the filter will be actually evaluated in the target OSI-SM agent; this is not the case in native CORBA environments.

Let us revisit first how filtering is supported in OSI-SM environments. Filter assertions on a particular attribute are evaluated by the attribute itself, while the ASN.1 compiler produces comparison methods. 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 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 and other comparisons. However, this requires the modification of both the IDL and the relevant programming language mappings. In summary, it is not easy to provide filtering in native CORBA environments as the mapping of IDL types to concrete language classes is not rich enough, and is lacking support for comparison.

C. Fine-Grain Event Dissemination and Multiple Object Access Through the Management Broker

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 an Event Processing (EP) object. Managed objects will get access to it through local means, e.g., the factories may pass its reference to MO's at creation time. MO's 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 EFD's and logs, and may instruct the latter to send the event or to log it as a log record accordingly. This is similar to the behavior prescribed in [12] and [13]. 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 EFD's and setting the *destination* attribute to contain either their name or object reference.

It should be noted that the OMG has recently complemented the event service [10] through a *Notification Service,* which supports dissemination based on event content filtering. Despite this, the proposed approach is more lightweight than the operation of a notification server in a network element. Compared to the more general case in which a notification server operates in a node associated to a domain of network elements, the proposed approach results in direct event dissemination between managed and manager objects, reducing latency and management traffic.

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 that they 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. Reference [23] proposes an enhanced model in which any relationship could be navigated, with filtering possibly applied at various stages of the object selection process. Other mechanisms may be invented in the future that best suit the needs of particular management environments.

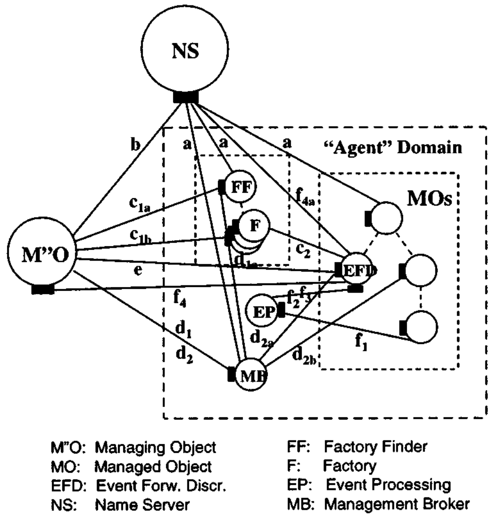

Fig. 5. A complete architecture for OSI-SM to CORBA mapping.

This is 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, an MB should always exist in an "agent" domain with its name bound to the domain naming context, e.g., $\{c = GB,\, o = UoS,\, ou = CCSR,\, cn = NM - OS,\, brokerId = CMIS\}$. 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 Fig. 5, including the event dissemination through EFD's. This updates the architecture presented in Fig. 3.

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., $\{c = GB,\, o = UoS,\, ou = CCSR,\, cn = NM - OS\}$. 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). An 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

```
// Scope_t, Filter_t and Sync_t map 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;
    i_ManagedObject     objectRef;
};

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);

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

    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       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);
    // . . .
};
```

Fig. 6. Code 3. Generic CMIS-like Managed Object Access in IDI.

MB. In the latter case, it will probably access more than one MO's, e.g., to suspend the operation of all its EFD's (interactions d_2, d_{2a}, and d_{2b}). An 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 of 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).

We present a CMIS-like broker below, 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. The reason for providing the same functionality through the CMIS management broker is that the latter may play the role of a CORBA-based management agent, with the managed objects being plain engineering objects, i.e., *without* individual IDL interfaces. Code 3 (Fig. 6) shows (a part of) the specification of the CMIS-like MB.

V. Design, Implementation, and Migration Aspects

Given the target CORBA-based framework that was depicted in Fig. 5, we will examine how this can be implemented. We

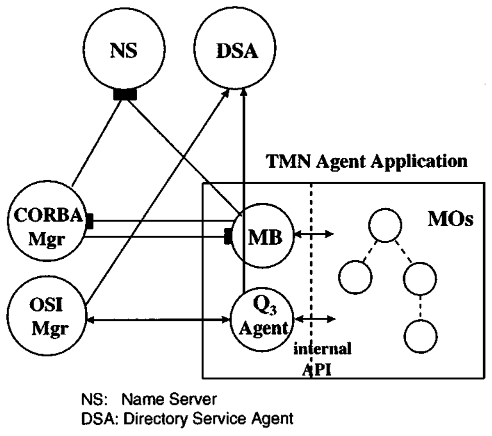

Fig. 7. Dual Q_3 and CORBA agent.

will consider, in particular, a phased approach which reuses initially parts of the OSI-SM based infrastructure. 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 toward 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 API's. The MB may be used in conjunction with the existing Q_3 agent object within an agent application. 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 Fig. 7 and has no impact at all on the implementation of managed objects which are based on TMN platform technology, e.g., OSIMIS [22]. 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 [10], while OSI manager objects get access to the presentation address of the OSI agent through the OSI directory [15]. 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. This architecture exploits the fact that the object models are the same in the two frameworks, and provides a "dual-agent" access paradigm. 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.

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. T. Tin of UCL, together with the author, first implemented a gateway version of the MB in 1996. This was the first CORBA-to-CMIS/P gateway that provided full OSI-SM functionality. The author subsequently designed and implemented the tightly coupled dual agent approach depicted in Fig. 7.

Implementing the gateway was fairly straightforward. The only difficulty encountered was the bidirectional 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 customized to produce the equivalent IDL types as well as the conversion routines in both directions. Implementing the tightly coupled dual agent version was also straightforward given the well-defined OSIMIS API's 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 relinking them with the management broker library.

The complete CORBA-based framework requires also that individual managed objects become computational constructs with IDL interfaces. The difficult aspects regarding this realization 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 methods 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." This is far from desirable since it requires handwritten routines for every attribute type. On the other hand it is the only approach to support filtering at present.

VI. Performance and Scalability Considerations

One of the key concerns often voiced regarding the use of distributed object technologies in telecommunications management environments is scalability. Managed network elements and TMN element and network management operations systems contain a very large number of managed objects. In addition, TMN OS's contain many "managing" objects. The question is whether it is feasible to expose all those objects to the distributed processing environment as separate CORBA objects. The advantage of doing this is strong typing and the use of standard CORBA API's. On the other hand, an object becomes a "first class citizen" of a distributed processing environment at a price, given the additional support required by the underlying infrastructure. The management broker concept copes well with the problem of scalability when used essentially as a management agent, as every agent application needs to expose only a single

CORBA interface. However, this approach implies that TMN platform API's are used for managed object development.

Having implemented the management broker approach in parallel with an existing Q_3 agent infrastructure, another consideration is to assess the performance difference and the packet sizes in the two approaches—one based on the Q_3 interface and the other on its precise equivalent in CORBA IDL. We conducted a number of experiments on a pair of Sun Sparc20 workstations running Solaris and connected to a lightly loaded Ethernet. The Q_3 protocol stack used operated over TCP/IP using the RFC1006 method as specified in [24], while the CORBA protocol used was the Internet Interoperability Protocol (IIOP) [25]. The OSIMIS research prototype was used for the Q_3 interface (OSIMIS) [22] as opposed to a commercial CORBA implementation which uses the Basic Object Adapter (BOA). Performance and scalability issues may be different for the emerging Portable Object Adapter (POA), which will be supported in the next generation of products.

An example managed object was used, supporting methods with various data types in ASN.1 which were mapped to the equivalent IDL types. While a detailed performance comparison could be the subject of a separate publication, the key conclusion was that the CORBA-based access times for method invocations through the management broker were faster by about 30% on average compared to CMIS/P-based managed object access. Performing the same operation to the managed object through a direct IDL interface resulted in 45% faster times. In addition, the initial bind operation through IIOP was about 60% faster than association establishment using the Q3 protocol stack.

The next experiment concerned the size of packets exchanged, which were measured at the TCP payload level using the Berkeley *tcpdump* program. Performing an echo operation of 1 byte to an object in the first level of the MIT incurred 72/88 byte packets for the request/response in the case of Q_3, 177/47 bytes in the case of the management broker, and 101/32 bytes in the case of a direct IDL interface. Performing an operation asynchronously through the management broker, which means that an Interoperable Object Reference (IOR) is exchanged, resulted in 409/97 byte packets. After many different measurements with different argument types, we came to the conclusion that IIOP generates a relatively large amount of traffic in comparison to Q_3, especially when IOR's and IDL any types are exchanged, since the latter convey CORBA type-code information.

The final experiment concerned the memory required for managed objects in both TMN and CORBA platforms. In the case of CORBA, the ORB which is typically running as a separate process requires 2.7 Mb at runtime. The size of a process containing a single server object and an associated factory object was about 2.6 Mb. The size of the equivalent OSI agent was 2.3 Mb, but the latter also contains functionality of name resolution, scoping, filtering event reporting, and logging. The key issue, however, is not the size of the infrastructure which is incurred once, but the data size of managed objects at runtime. After various experiments and memory size measurements, it became clear that the data size of a CORBA object is about 2–3 times more than the size of a GDMO object, depending on the data type of the attributes and the relevant values. In addition, the particular ORB used seemed to have a problem coping with very large amounts of managed objects.

These experiments confirmed the fact that IIOP is expected to perform better than Q_3 interfaces, yet generating larger amounts of management traffic. On the other hand, the experiments also confirmed the fact that the required memory size is bigger for native CORBA managed objects, which may result in scalability problems. This reinforces the management broker approach presented in this paper as an approach that benefits from the better performance of CORBA without the associated potential scalability problems. The native CORBA-based managed object approach could be the next step, with more mature, second- or third-generation ORB products coming to the market.

VII. Summary

Given the emergence of distributed object technologies, with CORBA being the representative open technology, in this paper we examined in detail how such technologies can be used as the basis for future TMN systems. We presented first 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 of this approach is that only few objects in each "agent" domain need to export their names to the name server, which avoids performance problems and reduces the management traffic required for managed object discovery.

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, with the drawback that 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 reconsider its attribute model in the future and add expressiveness similar to GDMO; this will solve these problems. An advantage of the presented approach is that managed objects are not required to have separate IDL interfaces, which helps scalability for network elements with a large number of managed objects. After the implementation of this approach, the existence of two equivalent CORBA- and OSI-SM-based TMN platforms led us to conduct comparative performance experiments. We examined response times, packet sizes, and run-time memory requirements, and we found CORBA response times better, packet sizes relatively larger, and memory requirements much higher.

We should finally answer the question of what is the architectural impact to the TMN if a distributed object technology such as CORBA is adopted. The answer is that there should be no impact at all. The TMN architecture and methodologies will remain the same. Interface specifications will be produced in GDMO, given the already large existing base of GDMO specifications and the fact that it can be seen as a general-purpose management information specification language. On the other hand, guidelines should be put in place in order to guarantee that generic translation to IDL is possible. In addition, the Q_3 and X interface profiles will be modified, including CORBA protocols as valid choices for TMN interfaces; work has already taken place for X while it is underway for Q_3. The use of CMIS/P-GDMO or GIOP-IDL will become an engineering issue for implementing the same specifications. Finally, an additional benefit of using CORBA is that TMN OS components could be distributed across different network nodes.

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 potentially better performance, but the fact that it may become the ubiquitous technology for future heterogeneous distributed systems.

References

[1] ITU-T Rec. M.3010, "Principles for a telecommunications management network (TMN),", Study Group IV, 1996.

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

[3] ITU-T Rec. X.900, Information Technology—Open Distributed Processing, *Basic Reference Model of Open Distributed Processing*, 1995.

[4] Object Management Group, "The common object request broker: Architecture and specification," CORBA, Version 2.0, 1995.

[5] ITU-T Rec. X.500, Information technology—Open Systems Interconnection, *The Directory: Overview of Concepts, Models and Service*, 1993.

[6] NMF-X/Open, Joint Inter-Domain Management (JIDM) Specifications, "Specification translation of SNMP SMI to CORBA IDL, GDMO/ASN.1 to CORBA IDL and IDL to GDMO/ASN.1,", 1995.

[7] ITU-T Rec. X.722, Information technology—Open Systems Interconnection, *Structure of Management Information—Guidelines for the Definition of Managed Objects*, 1992.

[8] ITU-T Rec. X.703, Information Technology—Open Systems Interconnection, *Open Distributed Management Architecture*, 1997.

[9] S. Proctor, "An ODP analysis of OSI systems management," in *Proc. TINA'92 Workshop*, Narita, Japan, Jan. 1992.

[10] Object Management Group, "CORBA Services: Common object services specification (COSS),", rev. ed., 1995.

[11] T. Rutt, Ed., "Comparison of the OSI systems management, OMG and Internet management object models," X/Open Joint Inter-Domain Management Task Force, Rep. NMF, 1994.

[12] ITU-T Rec. X.734, Information Technology—Open Systems Interconnection, *Systems Management Functions—Event Report Management Function*, 1992.

[13] ITU-T Rec. X.735, Information Technology—Open Systems Interconnection, *Systems Management Functions—Log Control Function*, 1992.

[14] ITU-T Rec. X.720, Information Technology—Open Systems Interconnection, *Structure of Management Information—Management Information Model*, 1991.

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

[16] ITU-T Rec. X.710, Information Technology—Open Systems Interconnection, *Common Management Information Service Definition (CMIS), Version 2*, 1991.

[17] L. A. de la Fuente, J. Pavon, and N. Singer, "Application of TINA-C architecture to management services," in *Toward a Pan-European Telecommunications Service Infrastructure*, H.-J. Kugler, A. Mullery, and N. Niebert, Eds. New York: Springer, 1994, pp. 273–284.

[18] S. Mazumdar, "Mapping of common management information services to OMG COSS specification," AT&T Bell Labs, TM#BL0112540-96.09.30-02, 1996.

[19] J. Hierro and J. Gonzalez, "Common facilities for systems management," Telefonica I+D Submission to the NMF—X/Open Joint Inter-Domain Management task force, 1996.

[20] G. Pavlou and D. Griffin, "Realizing TMN-like management services in TINA," in *J. Network Syst. Management (JNSM)*, 1997, vol. 5, pp. 437–457.

[21] E. Garcia-Lopez, "Distributed management facilities architecture,", TINA-C baseline doc. TB_EGL.002_2.1_1996, 1996.

[22] G. Pavlou, G. Knight, K. McCarthy, and S. Bhatti, "The OSIMIS platform: making OSI management simple," in *Integrated Network Management IV*, A. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. London, U.K.: Chapman & Hall, 1995, pp. 480–493.

[23] G. Pavlou, A. Liotta, P. Abbi, and S. Ceri, "CMIS/P++: Extensions to CMIS/P for increased expressiveness and efficiency in the manipulation of management information," *IEEE Network*, vol. 12, no. 5, pp. 10–20, 1998.

[24] ITU-T Rec. Q.811, Specifications of Signaling System No. 7, Q3 Interface, *Lower Layer Protocol Profiles for the Q3 Interface*, 1996.

[25] Object Management Group, "Internet inter-operability protocol (IIOP)," CORBA version 2.0, 1995.

George Pavlou received the Diploma in electrical engineering from the National Technical University of Athens, Greece, and the M.Sc. and Ph.D. degrees in computer science from University College London, U.K.

He is a Professor of information networking at the Center of Communication Systems Research (CCSR), School of Electronic Engineering and Information Technology, University of Surrey, U.K., where he leads the Networks Research Group. His research interests include protocol performance evaluation, network planning and dimensioning, traffic engineering and management of ATM and QoS-based Internet, programmable and active networking, multimedia service control, and distributed object-oriented platforms. He has contributed to standardization activities in ISO, ITU-T, NMF/TMF, OMG, and TINA.

Dr. Pavlou serves on the Editorial Board of the *Journal of Network and Systems Management (JNSM)* and the IEEE COMMUNICATIONS SURVEYS, and he is on the program committee of a number of international conferences on management and control.