OSI Management Model in the ODP Architectural Framework

Kazi Farooqui
Department of Computer Science,
University of Ottawa,
Ottawa K1N 6N5, Canada.
(Internet: farooqui@csi.uottawa.ca)

Abstract

The ODP model is a generic architectural framework for describing distributed systems. It is relevant and applicable to different application-domains. In this paper we demonstrate how the architectural concepts of the ODP model help in the identification of a distributed management model. It is the goal of this paper to position the current OSI management model within the ODP architectural framework. It is an attempt in answering the question: "how can ODP be used to enhance the OSI management model towards a truly distributed management model. As a first step in that direction, we show the application of ODP computational and engineering models in the OSI management domain. It is desired that the architecture for true distributed OSI management must gracefully evolve from (and be backwards compatible with) the current point-to-point architecture.

Keywords: Network management, OSI systems management, CMIS, CMIP, manager, agent, managed objects, open distributed processing, computational model, engineering model, distribution transparency.

1.0 Introduction

The premise of our work is the recognition of the inherently distributed nature of the management activity. OSI Systems Management [1] is essentially a distributed processing activity and, as such, should be expressible using the Reference Model for Open Distributed Processing (RM-ODP) [2,3,4,5].

The RM-ODP is a generic framework for the development of numerous future standards in various application domains. It may be seen as a *meta-standard* to coordinate and guide the development of domain specific ODP standards. This paper illustrates the application of ODP to the OSI management domain that would lead to the definition of an *open distributed management architecture.*

The RM-ODP gives a generic framework for describing a distributed system completely from different system abstractions - a framework of *viewpoints*. OSI-Systems Management offers a particular example of a distributed management system. The question is which aspects of OSI management needs to be described and in which viewpoint(s) these aspects should be placed. This paper presents the aspects of OSI-Systems Management related to the computational and engineering viewpoints.
Section 2 provides a brief introduction to the OSI management model and to the RM-ODP - a generic architectural framework for the organization of distributed systems. Section 3 presents the ODP computational model as an object-oriented framework for the structuring and organization of distributed applications. In section 4, we show how the full generality of the ODP computational model can be applied to the implicit and simple computational model of the existing OSI management that permits the modelling of distributed management applications. Section 5 presents the ODP engineering model as a framework for the definition of an object-based distributed platform for the support of interactions between distributed application components. Section 6 illustrates how the genericity of the ODP engineering model can serve as a basis for the identification of distributed platform for management applications. Conclusions are drawn in section 7.

2.0 Background

This section provides a brief introduction to the current OSI Management Model and to the generic ODP model.

2.1 OSI Systems Management Model

The world of OSI Management Model is populated with Managing Systems and Managed Systems. All (traceable) interactions in the OSI Management Model are between the managing and managed systems. As shown in figure 1,

the managing systems consists of *manager processes* and managed systems consists of *agent processes* and a set of *managed objects* organized (as a tree) in a *management information base* (MIB). Managed Objects (MOs) are an abstraction of the actual resources being managed (for the purpose of the management of the resource). MOs represent the manager's view of the resource. Systems management is achieved over associations established between manager processes (in the managing system) and agent processes (which represent and control a set of managed objects) in the managed systems

The manager process requests the execution of management operations from the agent which performs them (through the involvement of managed objects) and returns the responses to the manager. The agent may also generate event reports asynchronously to be sent to the manager. The manager-agent interactions are supported by an OSI standard management service known as the Common Management Information Service (CMIS) [6], implemented by the OSI standard management protocol called the Common Management Information Protocol (CMIP) [7]. The CMIS supports the following operations and notifications:

a. M-Get: operation is used by the manager to retrieve attribute values from managed objects.
b. M-Set: operation is used by the manager to set the attributes of the managed objects.
c. M-Action: facilitates non-standardized operations on managed objects.
d. M-Create: operation allows the manager to request the agent to create an instance of MO.
e. M-Delete: operations allows the manager to request the agent to delete an instance of MO.
f. M-Event-Report: notification is used by the agent to report to the manager the occurrence of an event associated with an MO.

2.2 ODP: A Generic Architecture for Distributed Systems

The Reference Model for Open Distributed Processing (RM-ODP) is an architectural framework for the integrated support of distribution, inter-working, inter-operability, and portability of distributed applications. It provides an object-oriented model for building open distributed systems[8, 9].

The ODP model is sufficiently general to be used in several areas. Specific fields of ODP applications include advanced telecommunication architectures such as Intelligent Networks, Automated Manufacturing Systems, Office Systems, Management Information Systems, etc. [10].

2.3 ODP: A Framework of System Abstractions

The ODP framework of viewpoints partitions the concerns to be addressed in the design of distributed systems. A viewpoint leads to a representation of the system with emphasis on a specific set of concerns. Different viewpoints address different concerns, but there is a common ground between them. These viewpoints should not be seen as architectural layers, but rather a different abstraction of the same system. An object-oriented approach has been adopted to model distributed applications in each viewpoint.

While all the viewpoints are relevant to the description of distributed systems, the computational and engineering viewpoints are the ones that bear most directly on the design and implementation of distributed systems. From a distributed software engineering point of view, the computational and engineering models are the most important; they reflect the (software) structure of distributed applications and of the underlying distributed platform more closely. In this paper we concentrate on the *computational modelling* and *engineering modelling* of the OSI Management Model.

3.0 ODP Computational Model: An Object-Oriented View of Distributed Application

The computational model offers a framework for describing the structure and organization of applications into distributable components (*computational objects*), identification of interfaces of the distributable components (*computational interfaces*), identification of interactions (*interrogations, announcements*, etc.) between (the interfaces of) application components, and for the expression of distribution requirements (*environment contract*), from the underlying distributed execution environment, for the support of interactions between application components.

The computational model is based on a *distributed-object model*. The model prescribes an object-oriented view of the distributed application. In the computational model a distributed application is viewed an 'object world' populated with concurrent (computational) objects interacting with each other, in a *distribution-transparent* abstraction, by invoking operations at their interfaces. An object can have multiple interfaces and these interfaces define the interactions that are possible with the object.

While the computational objects are the units of structure and encapsulation of (application-specific) services, interfaces are the units of provision of services; they are the places at which objects can interact and obtain or offer

services. A computational object may play different (application-specific) *roles* at its interfaces. These roles could be *client* or *server* role, *producer* or *consumer* role, *supervisor* or *subordinate* role, etc. A computational object may support multiple interfaces which need not be of the same type. Interfaces of the same type may be provided by objects which are not of the same type. Each object may provide interfaces which are unlike those provided by other objects [11].

4.0 Computational Model of OSI Management

In this section we show how the implicit and the simple computational model of the current point-to-point OSI management can be extended, using the 'generic' ODP computational model, to model the distributed OSI management scenario.

4.1 Implicit Computational Model of OSI Management

The current OSI management model implicitly possesses a very simple computational model. This corresponds to the point-to-point management model comprising of a *manager object* (in a managing system) interacting with a (set of) *managed object* (s) through an *agent object*[1](in a managed system), as shown in figure 1. This model is limited to management applications involving point-to-point communications. It does not model a fully distributed management scenario as portrayed by complex management applications.

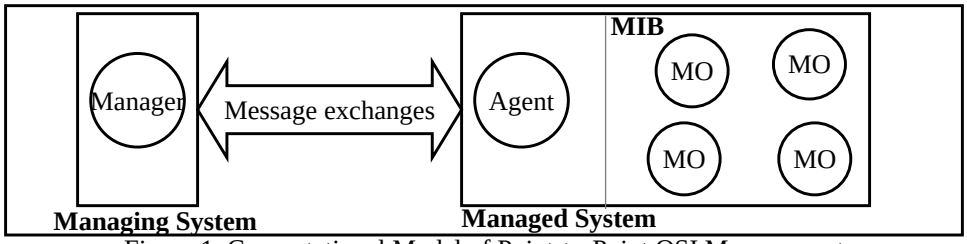

Figure 1. Computational Model of Point-to-Point OSI Management

4.2 Towards a Computational Model of Distributed OSI Management

OSI management applications are inherently distributed applications and the modelling of interactions between management application components should exploit the full genericity and power (concepts, structures, and rules) offered by the ODP computational model. This would enable the expression (and subsequent analysis) of complex distributed management applications through a single computational model.

We propose a *distributed management computational model* as an object world populated with a set of "*management objects*" distributed over different open systems and interacting in complex ways through operations invocations and notifications. These *management* objects may be *managing objects*, or *managed objects*, or may include both functionalities.

The section provides an interpretation of the existing OSI management model using the concepts of the ODP computational model and offers extensions towards a *distributed management computational model* (figure 2) within the framework of the 'generic' ODP computational model.

1. **Computational Objects**: The OSI management model comprises of objects either in the *manager* role or *managed*[2] role. However, these roles could be modelled through the interfaces of a computational object, as discussed below. Hence a generic *management* object serves as a computational object of the *distributed management computational model*. The *management* object may possess multiple interfaces, and it is these interfaces that may act either in the *manager* or *managed* role (or some application-specific role). The *management* object may (dynamically) offer either the *manager* interface(s) or *managed* interface(s) or both. The *management* objects serve as the unit of distribution and failure.

   The *management* computational object serves as an abstraction which is useful for modelling the relationship and collective behavior of the set of interfaces it supports. It offers a powerful modelling concept, and as discussed in

1. It will be shown later in section 6, that part of the agent functionality can be placed in the ODP engineering model.
2. A managed object is an abstraction of a real resource for the purpose of managing that resource.

section 6.3 it can be decomposed into implementable engineering objects of desired granularity.

2. **Computational Interfaces**: The ODP computational model prescribes that a computational object may contain both *operational interface* and *stream interface*. The OSI management model contains only operational interfaces. However, a *management* object may possess stream interface if it is desired (such as in multimedia applications) to sample the stream data locally at a *management* object.

3. **Role of Interacting Objects at Interfaces:** As mentioned earlier, the current OSI management model comprises of objects either in a *manager* or a *managed* role. The ODP computational model has fine a granular concept of *role* which can be applied to an interface of an object or can be associated with an individual interaction (operation) at an interface. The role taken by an object can vary at an interface or for a particular interaction at the interface. The ODP computational model has a *fundamental* concept of role: the computational interfaces model different interaction concerns of a computational object, for e.g., the *client* role (the initiator of an invocation), the *server* role (the responder of an invocation), *producer* role (originator of information flow) and *consumer* role (destination of information flow). In general, interactions between objects should be treated as taking any of these roles. In keeping with this genericity, it is flexible (as illustrated below) to model a *management* computational object as possessing *manager* role interfaces, *managed* role interfaces, or some application-specific interfaces (which includes the interface between managed object and the real resource).
   Interfaces in the *manager* role are *client* interfaces invoking operations on *managed* role interfaces acting as *server* interfaces.

4. **Computational Interaction**: The computational model (in addition to describing the activities within the *management* computational object, also) describes the interactions that occur between the computational interfaces in a distribution transparent way. Interactions between the interfaces (of computational objects) are expressed by means of operation exchanges that occur between the bound interfaces. An important exercise is the mapping of OSI management *operations* and *notifications* on the ODP computational model.
   The interactions, in the computational model of figure 2, may take one of the two forms:
   a. a management operation initiated by the *manager* interface and performed by the *managed* interface, resulting in a response being returned to the *manager* interface, or
   b. a notification sent from the *managed* interface to the *manager* interface without any reply.
   The former corresponds to the *interrogation* style and the latter to the *announcement* style of computational interaction.

   A managed object, as specified in the current OSI management model, can support both operations and notifications. This can be modelled in the *distributed management computational model* as a *management* object with two interfaces, both in the *managed* role, one to accept operations (emitted by the *manager* interface) and the other to emit notifications.

5. **Environment Contract of Management Interfaces**: The ODP computational model prescribes that the computational object templates[3] and computational interface templates may have associated environment contract which contains the information affecting the way in which the underlying distributed platform supports the computational interactions; this information constrains the type of distribution transparencies, choice of communication protocols, etc. that must be placed in the interaction path between interacting objects. The *manager* or *managed* interfaces may include environment contract stating distribution transparency requirement, communication quality of service such as throughput, delay, etc. For example a managed object may require its notifications to be delivered to the manager within certain maximum delay.

6. **Behavior at Management Interface**: The behavior at a *manager* or *managed* interface defines all possible ordering of operations and notifications that can be emitted or accepted at the interface. The behavior constitutes the protocol between a pair interacting interfaces. The behavior is specific to management application and may vary between a pair of interacting management interfaces. This is defined in the current OSI management model as the

3. A template is a object (interface) specification.

exchange of CMIP PDUs. However, as discussed in section 6.3, CMIP is a communication support mechanism for the exchange of computational operations and notifications and it is modelled in the engineering model.

7. **Binding of Management Interfaces**: The ODP computational model supports a type-checked (dynamic) binding between object interfaces. The binding between the *manager* and *managed* interfaces can be based on the ODP computational type compatibility. The OSI management model defines a binding between a pair of objects. It lacks the capability to model an arbitrary interaction pattern between manager and managed objects. However, this model can be extended through the concept of binding object defined in the ODP computational model. The binding object template can define complex interactions within a configuration of *manager* and *managed* interfaces.
   In the context of current OSI management model, the binding object concept allows the modelling of multicast of notifications issued by a managed object such as an event forwarding discriminator.

8. **Failure Model**: The current OSI management model lacks a *failure model*. There exists no way for the manager and managed objects to be informed about the underlying infrastructure failure.
   The ODP computational model supports the notification of infrastructure failures through response messages generated by the ODP engineering infrastructure. This way *management* objects can be informed about the failures in the underlying distributed platform.

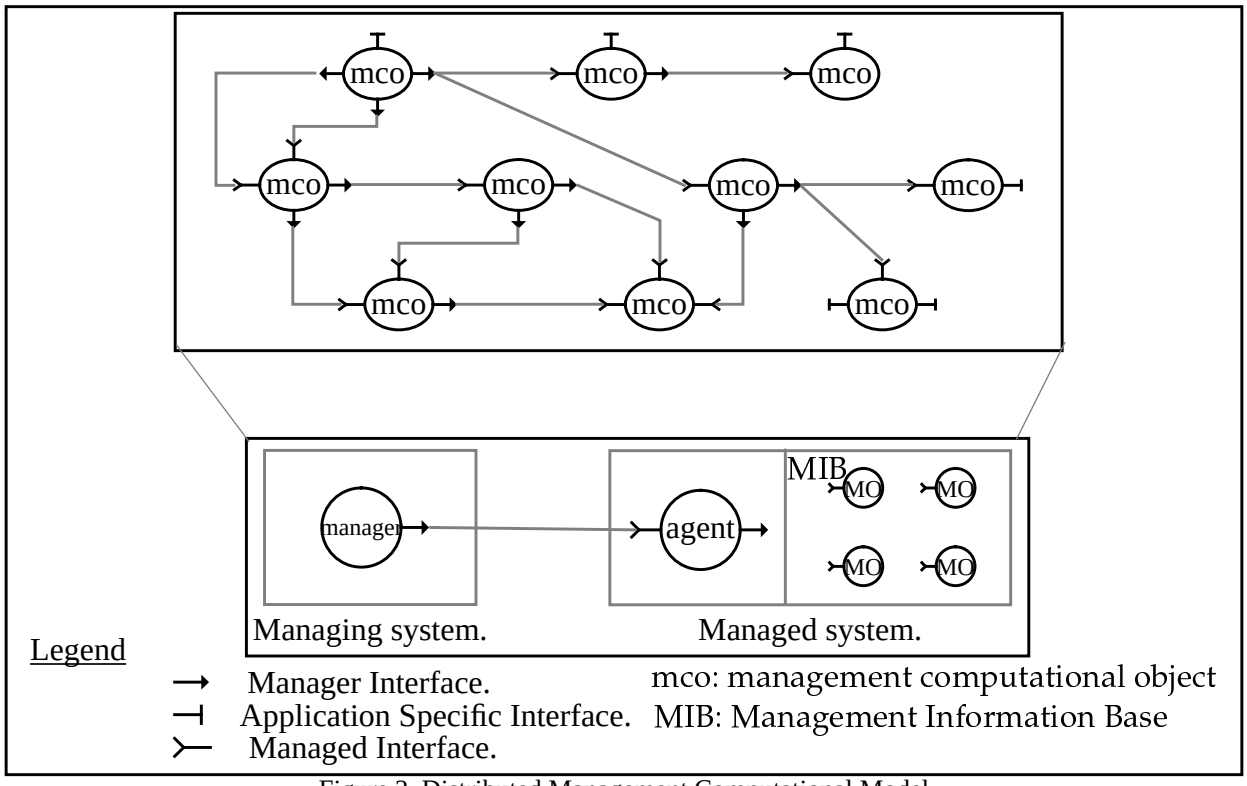

Figure 2. Distributed Management Computational Model

The *distributed management computational model* permits the modelling of the effect of interactions at one interface of the object on interactions at the other. A *management* computational object possesses multiple interfaces, each in either a *manager* or *managed* role (or some application-specific role). Management operations received on one interface would necessitate initiation of operations on other interface. A *management* computational object can be viewed as a composite object acting as a manager object over one association and as a managed object over the other.

The *distributed management computational model*, as identified above, offers the full flexibility of the ODP computational model that permits the modelling of different distributed management scenarios:

1. a *management* object possesses multiple interfaces all of which are *manager* interfaces: this models the case of a single manager object managing a set of managed objects distributed on different open systems. Each pair of manager-managed interface models a management association between the involved objects.

2. a *management* object possesses multiple interfaces all of which are *managed* interfaces: this models the case of a single managed object (event forwarding discriminator) sending notifications, via different interfaces, to a set of manager objects distributed on different open systems or the case of a single managed object (log object) responding to the queries of different manager objects, or a combination of both.
3. a *management* object possesses multiple interfaces some of which are *manager* interfaces and some are *managed* interfaces (and some *application-specific* interfaces): this models a very general management scenario in which the *management* object takes the role of both the manager and managed objects. This corresponds to the generic and distributed management scenario where there exists a hierarchy of managers managing a set of distributed managed objects and reporting the status to (and responding to the queries of their) superior managers. It permits the modelling of the delegation of responsibility from one manager to another manager for the purpose of assigning different management (sub) activities to lower level (manager) objects.
4. a *management* object dynamically offers (creates) *manager* and/or *managed* interfaces and withdraws (deletes) them during its lifetime: a managed object representing a mobile resource (such as a cellular phone) may move from one location to another very frequently. It is convenient (and economical) to manage that resource locally. A local *management* object may dynamically offer a *manager* interface to manage the mobile object and report the status periodically to the *management* object higher up in the hierarchy. When the mobile object moves again, the *manager* interface may be withdrawn.

As shown in figure 2, this *distributed management computational model* is backwards compatible with the existing point-to-point management model. In the managing system, the *management* objects reduces to a single-interfaced object with the *manager* role and in the managed system the *management* object reduces to an (agent) object with two interfaces: one in the *managed* role and the other in the *manager* role. (However, as discussed below, part of the agent's functionality includes the engineering functionality and is modelled in the ODP engineering viewpoint). The *managed* interface is bound to the *manager* interface (of the *management* object) in the managing system. The *manager* interface is bound to the managed object at the root of the MIB.

The management concepts in the present OSI management model that are visible in the ODP computational model are mangers, managed objects, interactions between them such as operations, notifications (and their arguments), and (as explained in section 6.4) part of the agent functionality. In the computational model, the role of the agent is as an interceptor of operations invoked by the manager. Some of the management operations invoked on the agent, such as M-CREATE, M-DELETE are performed by the ODP engineering functions as discussed in section 6.4. Other operations such as M-Get, M-Set, M-Action are performed by the managed objects.

5.0 ODP Engineering Model: An Object-Based Distributed Platform

The ODP engineering model is an architectural framework for the provision of an object-based distributed platform for the support of the distributed applications modelled in the computational model. The set of basic services and mechanisms, identified in the engineering model, are modelled as a collection of interacting objects which together provide support for the *realization* of (computational) interactions between distributed application components.

The engineering model *animates* the computational model. The model is concerned with *how* an application, specified in the computational model, may be *engineered* onto the distributed platform. The mechanisms which enable, regulate, and hide distribution (in the computational model) are identified in the engineering model. The engineering model provides a machine-independent execution environment for distributed applications.

The services and mechanisms currently identified in the engineering model are generic in nature and can support distribution requirements of applications in a broad range of enterprise domains (such as Telecommunications, Office Information systems, Computer Integrated Manufacturing, etc.). However, domain-specific distribution support functions will have to be defined in the domain-specific engineering models (which may be considered as the specializations of the ODP engineering model). It is our effort to define an engineering model corresponding to the *distributed management computational model* described in section 4.2.

6.0 Engineering Model of OSI Management

This section is motivated by the fact that we need to identify and define the functionality of the distributed platform for the support of distributed management applications within the framework of the ODP engineering model.

6.1 Implicit Engineering Model of OSI Management

The current OSI management model implicitly possesses a simple distribution support model comprising of CMIS/CMIP, OSI-TP, ACSE, and the lower layer OSI transport protocols.

This architecture is sufficient to support point-to-point interactions between the managing and managed systems. In the following section, we propose an extension of this model such as to meet the requirements of a distributed management scenario by identifying the required distribution support mechanisms and organizing them within the framework of the 'generic' ODP engineering model. In particular the ODP functions and distribution transparencies that are applicable to the distributed management domain are identified.

6.2 Towards an Engineering Model of Distributed OSI Management

Our approach towards the identification of the *distributed management engineering model* for the support of distributed management applications consists of:

1. Identifying the relationship between the (relevant) OSI management model entities and the corresponding ODP engineering model concepts. This permits an engineering interpretation of the management model.
2. Identifying the distribution support functionality from the existing OSI management model. As discussed below, the agent object (in the managed system) contains some distribution support functionality. We separate and identify different ODP (distribution support) functions out of the current agent functionality and organize them in the engineering framework.
3. A fully distributed management scenario would involve interaction between *management* objects distributed over different (more than two) open systems. Such a management activity would involve multi-cast or otherwise complex interaction scenario. Such general interaction patterns are supported within the generic engineering model. We identify the ODP functions and distribution transparencies which are required to support such a distributed management activity.

6.3 OSI Management Model in the ODP Engineering Language

The following is an interpretation of the ODP engineering model concepts in OSI management model:

1. Basic Engineering Object: Basic Engineering Objects (BEOs) are the run time representation of computational objects (obtained through compilation, interpretation, or through some other transformation of computational object). A management BEO corresponds to the (system representation of the) *management* computational object. As discussed in [12], computational objects with multiple interfaces may be split (mapped) into multiple engineering objects[4] (or merged into a single engineering object), but the interfaces are preserved in the transformation. Hence in a real open system, we may see a *management* (computational) object with multiple (*manager* or *managed*) interfaces split into multiple engineering objects offering the original *manager* or *managed* interfaces. The decomposed management engineering objects may possess either single or multiple (manager or managed) interfaces. Additionally, they may be enriched with extra interfaces to:
   a. to synchronize actions between decomposed objects in order to maintain consistent computational object state.
   b. interact with the (distribution support) objects in the channel (see below).

2. Cluster: The ODP cluster expresses the concept of a configuration of related basic engineering objects that should be grouped together on a single node (an open system). A cluster is a unit of distribution, storage, and migration. A cluster corresponds to a group of management engineering objects obtained from the decomposition of *management* computational object with multiple interfaces. It may be noted that a *management* computational object is also a unit of distribution and engineering objects obtained by decomposing a *management* computational object can communicate directly, whereas objects in different clusters interact through *channels*. So, the modelling of *management* computational object must ensure that it offers only those interfaces which must (or can) be co-located and moved together as a unit. The concepts of *management* computational object and cluster together offer a very powerful modelling paradigm in the management world:
   a. *coordination of distributed management activities*: a *management* computational object with multiple *manager* interfaces models the case of the management of a set of related resources (distributed on the network) such that the outcome of management operation on one resource affects the issue of management operation on the other (via another interface). As shown in figure 3, this composite *management* computational object can be

4. This transformation may involve object(s) providing synchronization between the component BEOs.

represented in the engineering model as a cluster of related engineering (manager) objects (synchronized via internal interfaces) accomplishing a common management function.

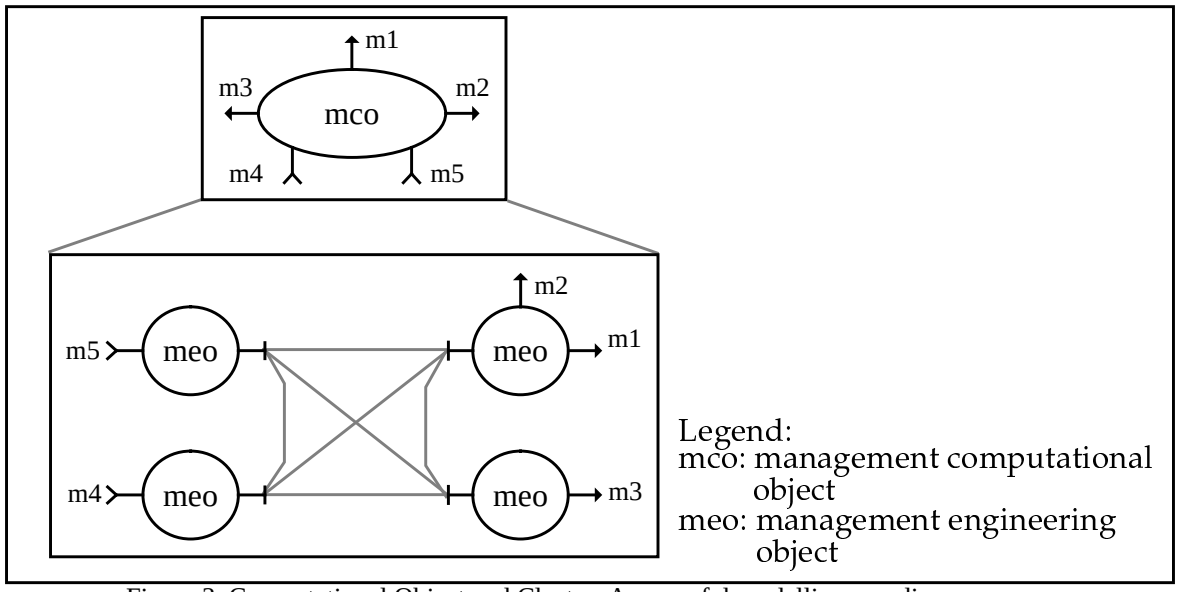

Figure 3. Computational Object and Cluster: A powerful modelling paradigm

b. *coordination of different management activities*: all managed objects corresponding to a mobile resource can be modelled (in the computational model) as *a management* computational object with multiple *managed* interfaces. In the engineering model, these interfaces would be represented as a cluster of engineering (managed) objects which are deactivated, migrated, and reactivated as a unit.

In the current OSI management model, a manager or a group of managed objects (i.e., MIB) represents a cluster.

3. Cluster Manager: A cluster manager performs management operations on the cluster such as activating, passivating, deleting, checkpointing, and migrating a cluster, etc. In the current OSI management model, part of this functionality is performed by the agent object. In a distributed management model, every managing/managed system will have a cluster manager per cluster of engineering objects obtained by decomposing a *management* computational object with multiple (*manager* or *managed* interfaces). It would be responsible for migrating, checkpointing, deleting the (related) group of engineering (manager or managed or combined) objects (obtained from *management* computational object), and hence also responsible for the provision of migration transparency, failure transparency, persistence transparency, etc.

4. Capsule: A capsule consists of a set of clusters and other distribution support objects such as transparency objects, protocol objects, etc. While a cluster corresponds to a group of engineering (manager or managed or combined) objects with a related management activity, a capsule corresponds to a set of clusters, in a given open system, with unrelated management responsibilities.

5. Capsule Manager: A capsule manager is responsible for the management of clusters in the capsule. The capsule manager functionality is currently a part of the agent functionality which is responsible for a group of unrelated managed objects.

6. Nucleus: A nucleus provides access to communications functions of an open system, to enable inter-system communication. In the current management model, the agent provides access to CMISE / CMIP communication facility.

7. Node: An open system containing both the *manager* and *managed* interfaces.

8. Channel: A channel is a configuration of *transparency stub objects*, *transparency binder objects*, and *protocol objects* providing a binding between a set of *manager* and *managed* interfaces. The structure (composition) of the

channel is dependent on the environment contracts of the (manager and managed) interfaces involved in the interaction. The channel is a composition of objects identified in section 6.5.

9. Stub Objects: The stub object add further interaction and /or information to interactions between interacting *management* computational objects to support (some aspect of) distribution transparency. As discussed in section 6.4, the stub functionality is currently a part of the agent functionality. It is required to transform management operations into messages that are exchanged as CMIP PDUs.

10.Binder Objects: The binder objects interact with one another to maintain the integrity of the binding between *management* computational objects. As discussed in section 6.5, the binder objects in the channel between manager and managed interfaces comprises of objects supporting location transparency, migration transparency, replication transparency. They support the management association between the involved interfaces, irrespective of whether the interfaces dynamically migrate or if they are replicated.

11.Protocol Object: A protocol object encapsulates communication protocol functionality for supporting communication between *manager* and *managed* interfaces. The CMIS/CMIP (and the underlying OSI stack of communication protocols) correspond to the protocol objects identified in the generic ODP engineering channel structure. The CMIS/CMIP assure that the *management* objects can interact remotely.

6.4 OSI Management Model: Extracting ODP Functions

The existing OSI management model implicitly possesses an engineering model sufficient to support point-to-point communication between management objects and functions (such as scoping, synchronization, etc.) to address a group of managed objects in the same system. The engineering support for the distribution of manager/managed objects on different open systems is not available. The computational and engineering functions are intermixed in the current model. Moreover, some of the engineering functions, such as those clubbed in the agent object, need to be identified, separated, extended in their scope for the purpose of distributed management, and organized within the framework of ODP engineering model. In the following we show how the *agent* functionality can be decomposed and performed by different ODP functions (and in a broader, i.e., distributed context). Such functions are identified and discussed below.

1. Scoping: The scoping allows a manager to select either a single managed object or multiple managed objects within a subtree of MIB for a specific operation. Basically, what is being achieved is that a group of managed objects in a single open system is being selected for a particular operation. (Additionally, these managed objects are organized as a tree).

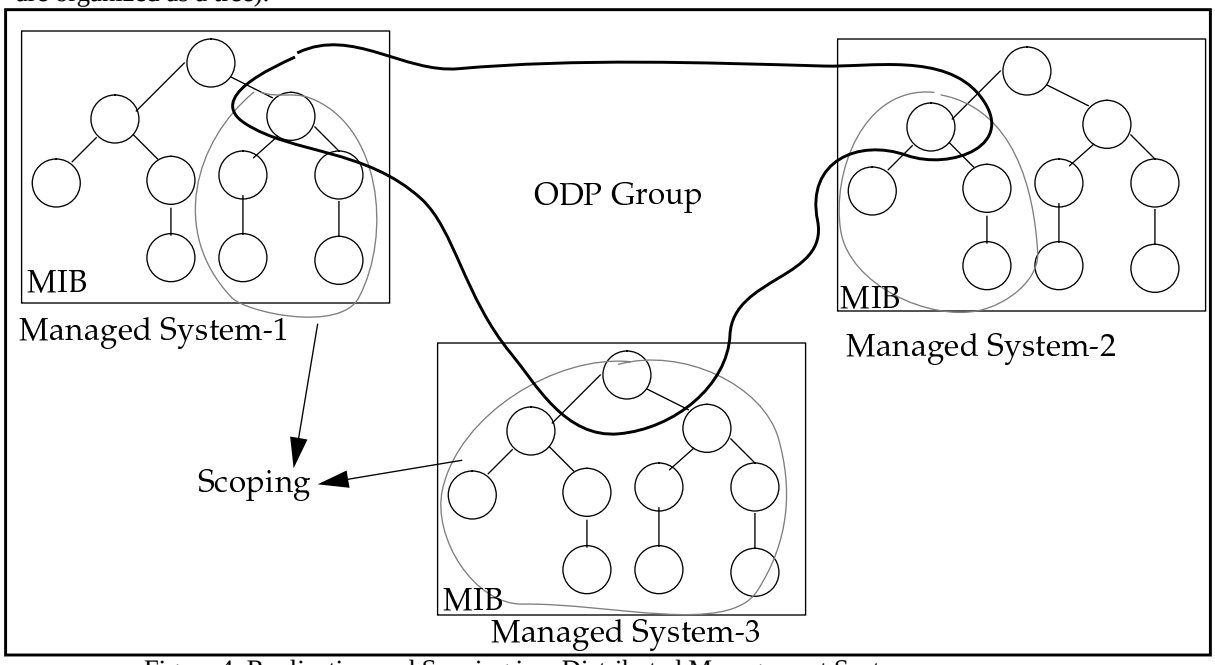

Figure 4. Replication and Scoping in a Distributed Management System

In a distributed management scenario, the concept of *scoping* can be extended using the ODP concept of *group*. In the distributed management model, a manager would require an operation to be performed on a *group* of managed objects distributed over different open systems. In general, these managed objects need not be organized as a tree. The related managed objects on which some common operations can be invoked are collectively identified by a *group-id*. The members of the group, identified by the *group-id*, are distributed on different systems. The scoping functionality can be taken out of the agent and replaced (or more precisely extended) with the ODP Group (Replication) Function.

As shown in figure 4, while replication allows the addressing and selection of a distributed managed objects (i.e., the base level objects) in different open systems, scoping allows the selection of managed objects in a local system.

The infrastructure first performs the multicast of the management operation on base level managed objects and then a local system function performs scoping on the local managed objects (which may be organized as a tree in a given open system).

2. Filtering: A filter specifies a condition that must be true in an object for it to be selected for a specified operation. Filtering imposes further constraints on the objects selected by scoping for an operation. While scoping uses the object hierarchy in an MIB to select objects, filtering uses the state of an object instance. Filtering can be performed by the managed objects before the execution of operation.

3. Synchronization: The agent performs two types of synchronization across managed objects in an open system: *atomic* and *best effort*. Atomic synchronization has all-or-none semantics, i.e., the management operation succeeds or fails on all objects selected. Best effort aims at performing the operation on as many selected objects as possible.

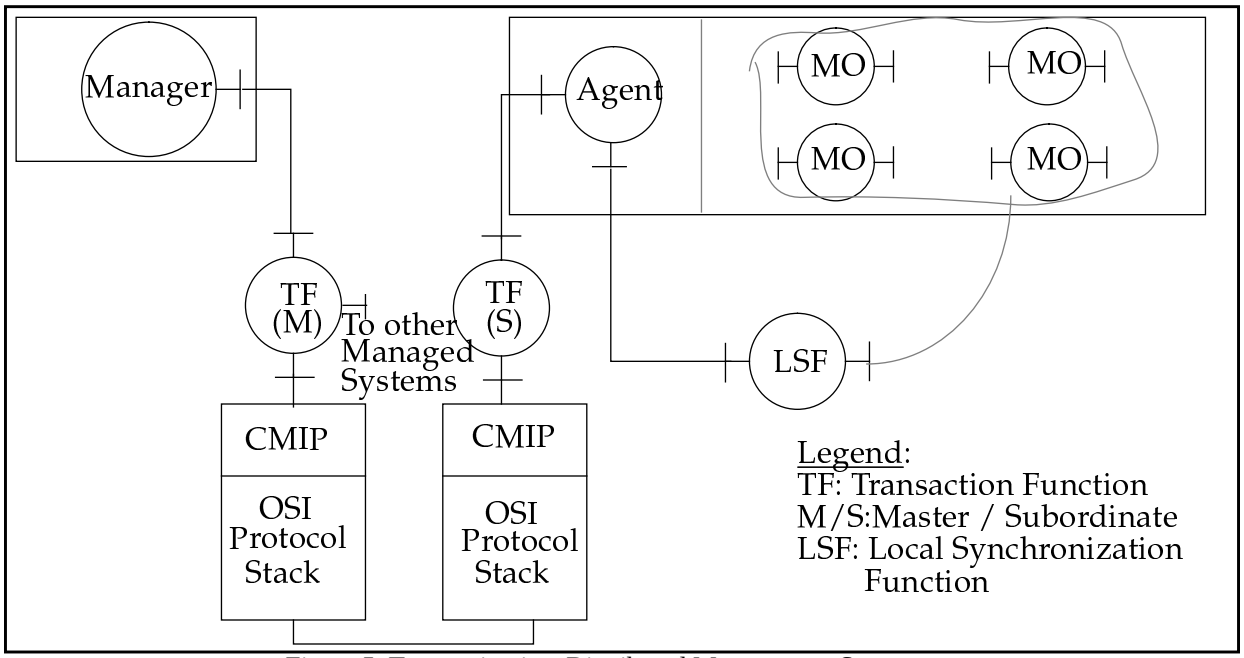

Figure 5. Transaction in a Distributed Management System

Again, as in the case of scoping, the scope of synchronization in the OSI management model is limited to objects in a single system. In a distributed management model, the manager would require a management operation performed *atomically* on a set of objects distributed over different systems. This can be achieved through the use of ODP Transaction Function.

A transaction model for distributed management systems is shown in figure 5. The management operations are intercepted by the Transaction Function. The transaction protocol[5] is actually carried over the CMIS service (supported by the CMIP protocol). One such CMIS service primitive that can be used for the exchange of transaction protocol is M-Action. Depending upon whether a Ready-To-Commit is obtained (enveloped in M-Action PDU) from all subordinate Transaction Functions (on the managed object side) or a Refuse-To-Commit is obtained from

at least one, the master Transaction Function (on the manager side) can issue a Commit or Abort (which is again carried by M-Action PDU). On the receipt of Commit (or Abort) the subordinate Transaction Functions can advise the managed objects (possibly via agent[6]) to make the results of the management operation permanent (or discard the results). The Local Synchronization Function, shown in figure 5, performs the transaction on the set of local objects.

4. Event Notification: In the OSI management model, the agent reports to the manager the occurrence of events associated with the managed objects. However, event notification function is a very commonly used function in the management world.
In order to relieve agent objects regarding the complex policies concerning event notification, this function can be readily mapped onto the ODP Event Notification Function. Event producers are managed objects and event consumers are managers. Both the managed objects and managers subscribe to the ODP Event Notification Function to send and receive events. Event Notification Function on the managed objects' open system[7] interfaces with the CMIP protocol object to send events via M-Event-Report PDU to the manager.

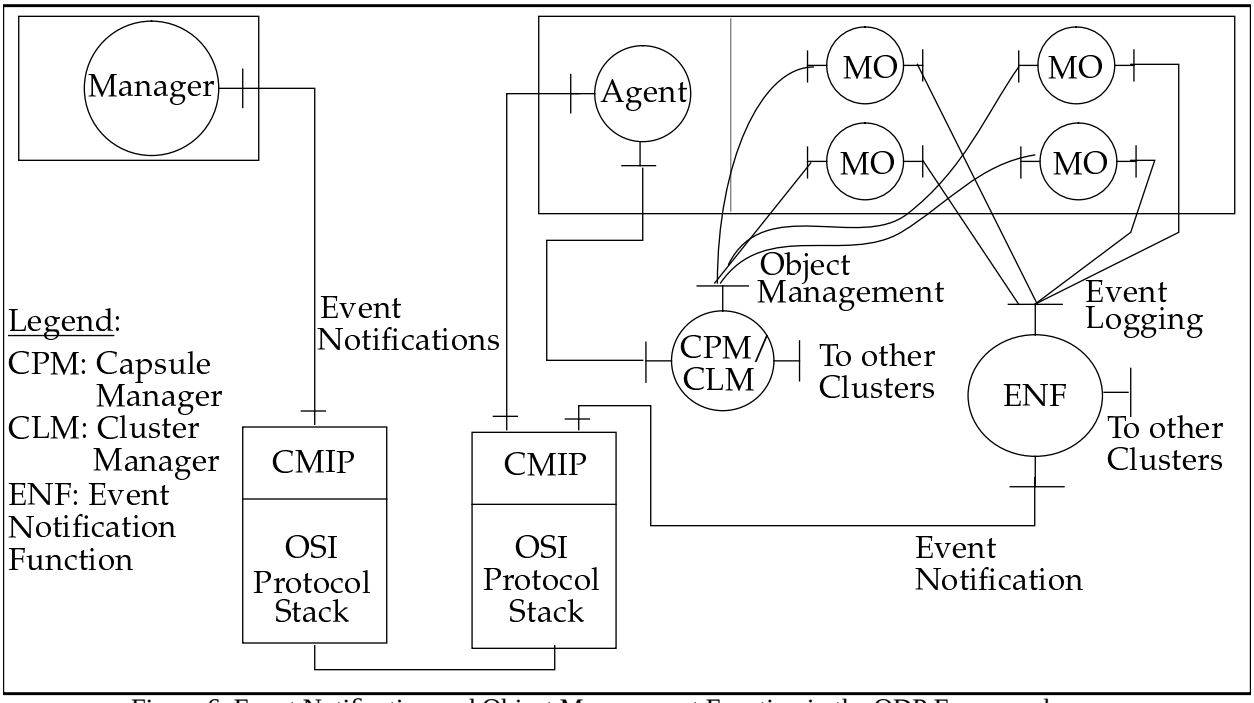

Figure 6. Event Notification and Object Management Function in the ODP Framework

5. Object Management Function: In the OSI management model, the agent is responsible for the creation and deletion of instances of managed objects. However, as discussed in section 4.2, the ODP cluster manager and capsule manager perform object management functions, which includes not only object creation and deletion, but also checkpointing, deactivation and reactivation. Hence, the managed objects in the cluster interface with the cluster manager/capsule manager to obtain object management services. As usual agent intercepts all management operations, and object management operations such as M-Create and M-Delete are performed by capsule manager/cluster manager.

---

5. The transaction protocol is essentially the exchange of operations such as Begin-Transaction, Ready-To-Commit, Refuse-To-Comit, Commit, Abort, and End-Transaction between master transaction function and subordinate transaction functions.
6. The routing of operation invocations on managed objects via agent is retained for backward compatibility with the current management model. Since the ODP functions such as Transaction Function, Replication Function, etc. perform the agent functionality, the indirection via agent can be eliminated.
7. Event Notification Function could exit on a system other than that of a manager or managed objects.

6.5 ODP Distribution Transparencies applied to OSI Management Model

The basic premise of our work is the recognition that management applications are inherently distributed applications. Given that the advanced telecommunications involve mobile objects and that these objects are also subject to management operations, the management model should evolve to address a distributed and dynamic scenario.

The details of distribution can be hidden from the management applications (specified in the computational model) through the *selective* application ODP transparency recipe. This section investigates which ODP distribution transparencies are applicable in distributed management environment.

1. **Access Transparency**: Access transparency hides the differences in data representation and invocation mechanisms from the interacting objects. The manager may have a data representation and invocation mechanism different from the objects it manages. In the management model, access transparency is achieved by requiring the application to invoke the CMIS service through standardized service primitives. Similarly the conversion of the CMIP PDUs to service primitives (operation invocations) on the manager/agent objects is done by the CMIP service interface. Consequently, this part of the protocol can be identified with the access stub.

2. **Location Transparency**: Location transparency hides from the client objects (interfaces) the location of the server objects (interfaces) in the distributed system. Location transparency requires that management objects (both managers and managed objects) have location-independent globally unique names.
   In dynamic environments, wherein managers wish to manage a particular managed object (resource) rather than the managed object on a particular system, location transparency is required.
   The naming structure assumes an important role w.r.t. location transparency. The naming structure in the current model follows an X.500 [13] naming scheme which is very location specific. In the management model, the provision of location transparency requires the alteration of the naming model to allow objects to retain their names irrespective of their migration in the distributed system.
   The ODP Relocator Object plays a central role in provision of location transparency and is a potential candidate for inclusion in the distributed management model.

3. **Migration Transparency**: Given that the managed objects are an abstraction of the real managed resource, and in a mobile environment such as in a telecommunication domain managed resources may dynamically move from one location to another, the manager object needs to maintain the management session established with the mobile managed object. The need for the provision of migration transparency to both the manager and managed objects is very obvious in highly dynamic mobile environments.

4. **Transaction Transparency**: The transaction mechanism involves coordination of activities among a configuration of objects to achieve (the specified) consistency. In most cases the condition that has to be satisfied is atomicity, i.e, all-or-nothing semantics. The case of transaction mechanisms arises in the management world when we adopt a distributed perspective of management activities. This requirement is illustrated in many management applications (for example, the Test Management Function [14]) in a distributed scenario. The operations such as Suspend_Test(), Resume_Test(), Set_Attributes(), are requested by the manager to be performed either on all the managed objects distributed on different systems or on none of them. In such cases the manager requires transaction transparency from the underlying distributed platform.

5. **Replication Transparency**: The need for replication transparency in the management model arise in a distributed management environment where proxies of managed objects may be kept (stored) at different management systems. This contrasts with the current management model which permits communication between two open systems.
   Manager wishes its operation, such as Get_Attributes(), or Start_Test(), to be performed on a set of replicated Test Objects (without invoking the identical operation on every individual object), organized as an ODP Group. Similarly replication transparency may be required by a managed object that wishes to send notifications (status reports) to a group of manager objects.

6. **Persistence Transparency**: Both the manager and managed object may require persistence transparency in order to reactivate them when a notification / operation is invoked on them. However, the case of reactivation at a different system needs consideration.

7. Conclusion

The ODP model is primarily a powerful and generalized model and like all powerful models the trade-off is in terms of simplicity. Being a generic model, it can be mapped onto different application-domains. In this paper, we have demonstrated the application of ODP architecture to OSI management model.

This paper is an attempt in answering the questions: how does OSI management model fit in the ODP framework and does the OSI management model be extended to a truly distributed management model using the ODP functions. We have demonstrated these complimentary aspects within the framework of ODP computational and engineering model.

8. References

1. C. Ashford. The OSI-Managed Object Model, Proceedings of European Conference on Object-Oriented Programming, Kaiserlautern, Germany, (Lecture Notes in Computer Science 707), Springer Verlag, Berlin, 1993.
2. Draft Recommendation ITU-T X.901 / ISO 10746-1: Basic Reference Model of Open Distributed Processing - Part-1: Overview.
3. Draft International Standard ITU-T X.902 / ISO 10746-2: Basic Reference Model of Open Distributed Processing - Part-2: Descriptive Model.
4. Draft International Standard ITU-T X.903 / ISO 10746-3: Basic Reference Model of Open Distributed Processing - Part-3: Prescriptive Model.
5. Draft Recommendation ITU-T X.904 / ISO 10746-4: Basic Reference Model of Open Distributed Processing - Part-4: Architectural Semantics.
6. Information Technology - Open Systems Interconnection - Common Management Information Service - ISO/IEC-9595, 1991.
7. Information Technology - Open Systems Interconnection - Common Management Information Protocol - ISO/IEC-9596, 1991.
8. K. Raymond. Reference Model of Open Distributed Processing: A Tutorial, Proceedings of the IFIP TC6 International Conference on Open Distributed Processing (eds: J.d. Meer, B. Mahr, O.Spaniol), Berlin, Germany, September 1993 (North Holland).
9. P.F. Linington. Introduction to Open Distributed Processing Basic Reference Model, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
10. G.Bregant. Towards a Convergence between Telecommunications Services Architectures and Open Distributed Processing, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
11. ANSA Reference Manual, Volume A., Release 01.01, Advanced Projects Management Limited, Cambridge, U.K., July 1989.
12. K.Farooqui, L.Logrippo. Viewpoint Transformation, Proceedings of the IFIP TC6 International Conference on Open Distributed Processing (eds: J.d. Meer, B. Mahr, O.Spaniol), Berlin, Germany, September 1993 (North Holland).
13. CCITT Recommendation X.500 - Directory Services, 1992.
14. CCITT Recommendation X.745 (1992) | ISO/IEC 10164-12: 1992, Information Technology - Open Systems Interconnection- Systems Management: Test Management Function.