MANAGEMENT FOR OPEN DISTRIBUTED PROCESSING

Morris Sloman

Imperial College of Science Technology and Medicine,
Department of Computing, 180 Queen's Gate, London SW7 2BZ
Email: mss@doc.ic.ac.uk

ABSTRACT

This paper explains what is management of a distributed system and why it is needed in an Open Distributed Processing (ODP) environment. An ODP environment is characterised by very large numbers of computer systems which interact to support a diverse range of applications. The computers and the applications are owned and managed by a number of different organisations so there is no single authority over all the entities within a typical ODP system. The paper defines the basic management concept of a managed object. It introduces the domain as a means of grouping objects to specify management policy and structure the overall management to cope with the complexity of an ODP environment. The paper relates management to the five viewpoints (enterprise, information, computation, engineering and technology) defined within the ODP reference model. Problems in using the Open Systems Interconnection Management Framework for distributed systems are also discussed.

1. OPEN DISTRIBUTED PROCESSING

The work on defining a reference model for Open Distributed Processing (ODP) is going on in ISO/IEC JTC1 SC21/7. It is likely that this will be merged with the work on a Distributed Application Framework (DAF) being undertaken by CCITT Q19/VII.

A computer system can be considered to perform three functions which require distribution in an ODP environment namely storage, processing and user access. ODP refers to these as *requiring aspects* [ISO SC21/7 89b]. ODP have also identified a set of *enabling aspects* needed to support distribution of processing, namely separation (communication), identification (naming), management and security. These services are needed to build practical ODP systems.

In order to simplify the complexity of describing and modelling distributed systems, the current ODP reference model has defined five viewpoints (Enterprise, Information, Computation, Engineering and Technology) to represent a system from a different set of abstractions [ISO SC21/7 90b].

This paper discusses the management concepts needed for the ODP reference model and how they relate to the ODP viewpoints. We also discuss the relevance of the ISO Open Systems Interconnection (OSI) standards being developed for management of communication systems.

Both the ODP and OSI work is based on object oriented concepts and so we use the term *object* as an entity within an ODP system. This object could be a person, user role, process or hardware component.

In section 2 we identify what management functions are needed in an ODP system and how management is affected by distribution. Section 3 defines the concepts of a managed object, manager, and management policy. The use of domains to group objects and structure management is also explained. Section 4 discusses management from the ODP viewpoints, concentrating on the computation and engineering viewpoints. Finally we give a critical appraisal of the problems of both the ODP and OSI approaches for management of distributed systems.

2. What is Management

A distributed processing application may span the computer systems belonging to a number of different organisations. There is no central authority in such systems and the management of the application as well as the underlying communication and processing resources will have to operate across organisational and legislative boundaries.

Management in an Open Distributed Processing environment is responsible for performing the long term planning, supervisory and control activities required to permit an ODP system to meet its specification and to evolve to incorporate new functionality [Sloman 88].

Managers are responsible for obtaining information about the activities and current state of objects within an ODP system, making decisions according to some overall management policy and performing control actions to change the behaviour of the objects.

An object within an ODP environment has a normal functionality which is defined by the particular service it provides and a management functionality which permits its normal activity to be monitored and possibly modified. Deciding what is management and what is part of the normal functionality of an object may be rather difficult. One criterion is that management of the object should not be critical to its normal operation. For example, a Transport Layer entity's normal functionality is to set up and clear connections, send and receive messages whereas management would include monitoring its performance and changing the number of buffers it uses. Similarly encryption would be considered a normal security service but distributing the keys used for encryption would be a management function.

In this paper we will concentrate on generic management functions common to many services within an ODP system and will not consider the operating system specific functions of processor and memory resource management which are well understood and can be considered part of the underlying support environment.

Management Functions

The OSI management framework [ISO SC21/4 89a, Klerer 88] identifies 5 functions for management of communications systems - namely configuration, fault, performance, accounting and security management. We contend that there is an

0-8186-2088-9/90/0000/0533/$01.00 © 1990 IEEE

alternative functional classification of management more suitable for distributed systems.

The assumption is that a distributed processing system will consist of a number of distributed applications which make use of underlying support services similar to those provided by a traditional operating system in a centralised system. Common services are processing, file storage, user access and communications but other typical services include directory or name mapping, trading, security and electronic mail. All these services, as well as the distributed processing applications need to be managed. Some of the management functions needed will depend on the service or application but the following set of management functions are common to many services and applications.

Configuration Management

It is necessary to specify an initial configuration of the hardware or software components which will form a service or application. The term configuration management is commonly used to mean only software version control but we use it to include specifying what classes of components are required, the instances needed, the interconnection or binding of interfaces and the allocation of software to hardware. In addition dynamic change of a system is required to permit evolution to incorporate new functionality or new technology.

Quality of Service (QOS) Management

A user of a service requires some pre-defined quality of service which can be in terms of delay, throughput, security level, undetected error rate, etc. Thus managing the QOS requires a means of specifying the required QOS, maintaining required QOS, notification of changes to QOS if it cannot be maintained and negotiating a different quality if the required one cannot be provided. Maintaining QOS includes both performance and fault management so it is a generalisation of these OSI functions. Note it may also be necessary to perform reconfiguration to maintain the QOS which indicates the management functions are inter-related.

Accounting

Accounting permits users to obtain information on usage of resources and is needed to allow suppliers of services to charge for the use of those services. It is also needed for fair allocation of resources to prevent a user from monopolising available resources within a service.

Monitoring

Monitoring of state, errors, performance and usage information is needed in order to support the above management functions, however the OSI framework does not define this as a management function.

Defining Policy

An important function of management is to define a management policy which governs how a system is managed and hence how it behaves (see section 3.3.)

Accounting is included in the the above classification, as most services need internal accounting for management purposes even if users are not billed for the service. Security is not classed as a management function but management interactions must be subject to strong security control. Authentication and access control are essential to prevent unauthorised users or managers from performing management actions on components of a service or application. Thus all services in a distributed environment need the use of a security service just as they need the use of a communications service, but security management is not a generic management service required for managing other services.

The generic functions needed for management of entities within a distributed system are configuration management, QOS management, accounting, monitoring and facilities for defining policy. These are the management functions which must be supported in addition to the normal functionality defined by the service or application.

3 KEY CONCEPTS

3.1 Managed Objects and Managers

A managed object is an entity to which a management policy applies and whose behaviour can be monitored and/or changed by a manager. A managed object could be a hardware or software component, a collection of information (e.g. a data structure) or even a person.

Managers monitor the activities of managed objects, make management decisions based on that information and perform control actions on the managed objects. A manager may be human or automated. Managers may also be managed objects for higher level managers.

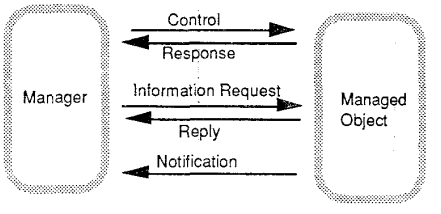

Fig. 3.1 Management Interactions

Three types of *interactions* between managers and managed objects can be identified, as shown in fig. 3.1. Control actions are directed by the manager object to the managed object, requests for information by the managing object result in a reply from the managed object, and unsolicited notifications from the managed object are sent to the manager to report faults or events.

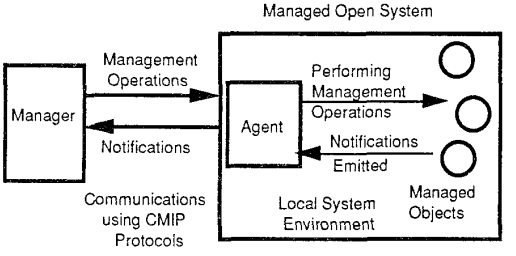

Fig. 3.2 OSI Interaction Model

The OSI Management framework defines the Common Management Information Services (CMIS) and the Common Management Information Protocol (CMIP) to implement the services. CMIS essentially defines the same interactions as those shown in fig. 3.1. However the OSI management model introduces an Agent between the Manager and Managed Objects, as shown in fig. 3.2. The reason is that the OSI

Reference Model insists on standardisation of peer-to-peer protocols and CMIP was defined before the model to which it relates. A manager obviously does not have a peer-to-peer relationship with a managed object, so an agent was artificially introduced in the model. The agent can also take on the role of manager with respect to other agents.

Our view is that the agent should not be in the interaction model. There may be implementation or policy reasons for a hierarchical management structure, with a local system manager which appears as a managed object to the remote manager (see section 4.4.1), but this should be an implementation option.

3.2 Domains

There is a need for multiple coexisting management views and boundaries of responsibility, each based on different structuring criteria, within a large distributed processing environment. In order to cope with the complexity of management of large distributed processing systems it is essential to provide a common framework for partitioning the overall management to reflect these different views. Domains provide a flexible and pragmatic means of specifying boundaries of management responsibility and authority. They permit a group of managed objects to be treated as a single entity for the purposes of specifying policy, which provides the basis for coping with the complexity of large scale distributed systems. However, it is necessary to permit different management domains, with members in common, to coexist in order to reflect the different required views of management.

A *domain* is an object which represents a collection of managed objects which have been explicitly grouped together for a purpose i.e. *to which a common management policy applies*. The objects may be resources, workstations, modems, processes, etc, depending on the purpose for which a particular domain is defined. A Domain has an attribute called an *Object Set*, which is the set of identities of member objects. The minimum representation of an object in a domain is a unique identifier which can be used to locate the object by means of a location service, but the domain may optionally hold a local name for use by human managers to refer to an object. (Note an implementation may also choose to cache managed object addresses).

An object is referred to as a *member* of a domain if its identity is a member of the domain's Object Set. Objects may be a member of more than one domain at a time. Domains are persistent even if they do not contain any objects - it must be possible to create an empty domain and later include objects in it. Since domains are themselves objects, they may be members of other domains (see section 3.4).

For an object to be manageable, it must exist within a domain, so objects should always be created within a domain.

Domains do not encapsulate the objects themselves - managers or external objects may interact directly with an object in a domain.

3.3 Management Policy

Domains provide the means for specifying management policy for a group of managed objects rather than having to do this on an individual object basis. The overall management objective and external constraints relating to laws (e.g. the Data Protection Act), regulations or higher level are two aspects of policy for a domain which would be difficult to specify formally.

Internal constraints place restrictions on the operations which can be performed on objects in a domain. These can be expressed declaratively in terms of domain or object attributes or as obligations which specify what the object must do to be a member of the domain. For example, an object must support a minimum interface in order to be included in the domain or it may be a policy decision to permit only a single manager in a manager domain to prevent problems of multiple managers. We anticipate using a logic based language for specifying internal contraints with automated checking.

Access rules specify what management operations the managers may perform on the objects they manage[Moffett 90] and are also an aspect of domain policy. An access rule is a ternary relationship between a domain of managers, a domain of managed objects and a set of permitted operations. For example, all members of the domain SysProgrammers are allowed to start and and stop the objects in the DepartmentServices domain. The permitted operations are a subset of the management operations defined by the interfaces to the objects in the domain. There can be multiple manager domains, each with an access rule relating to a single domain of managed objects.

3.4 Domain Relationships

There is a set of domain relationships which can be used to model management structures.

Two domains are defined to be *disjoint* if their object sets are disjoint.

Two domains *overlap* if there are objects which are members of both domains. An example is the shared management of a gateway interconnecting two networks by the management centres of each network. This can be accomplished by including an object from one domain into another. A special case of overlapping occurs when the objects in one domain are a *subset* of the objects in another.

Implicit overlap may occur between two domains containing managed objects of different type but referring to the same real-world entity. An example is scheduling and maintenance domains in which putting a workstation out of service in the maintenance domain makes it unavailable in the scheduling domain. Implicit overlap is likely to occur where there is a functional partitioning of management into different domains.

The main method of structuring management is to delegate authority over a set of objects to another manager. This can be accomplished by a *subdomain*, within another domain. A subdomain is a managed object which is a domain and can contain other domains or managed objects. It enables a different management policy to be applied to the subdomain compared with that of the enclosing domain. Delegation of authority is accomplished by setting up a new access rule permitting a different manager (domain) to manage the subdomain.

4 ODP VIEWPOINTS ON MANAGEMENT

We now discuss how the above management concepts can be related to the ODP viewpoints.

4.1 Enterprise Viewpoint

The enterprise viewpoint is concerned with identifying management structure and policy both within an organisation and between interacting organisations.

The enterprise viewpoint can be modelled in terms of objects representing user roles, business and management policies, the system and its environment. Overall management policy specifies objectives and goals of individual organisations and the inter-organisation interactions. There is also the policy relating to evolution and change – how and by whom it is instigated.

It is important to identify management roles in terms of the function and the capability of the role rather than the person who fulfils the role. The manager objects are typically roles such as accounts manager, departmental manager, personnel manager, plant operators. Managed objects would include information, such as stock control, accounting, orders; equipment ranging from a chemical plant to factory machinery; as well as information transformations required due to the nature of the enterprise. The access rules would identify the relationship between manager roles and managed enterprise objects in terms of responsibility for defining policy rather than performing specific management operations.

Domains reflect organisational structuring within an enterprise and the inter-organisational relationships. Defining what organisational domains exist and what policies should be applied to them is part of this viewpoint. Delegation of authority and partitioning of responsibility is specified by refining domains into subdomains and allocating managers to subdomains.

A manager domain corresponds to a role whose management responsibilities can be specified independently of the manager instance (person) who holds the role at a particular time. It also permits multiple managers to perform the same role simultaneously c.f. a personnel management department which may contain multiple managers.

4.2 Information Viewpoint

The information viewpoint is concerned with identifying the management information required to permit managers to perform their function. There is no distinction made between automated or manual parts of the system in this viewpoint. Defining management information involves identifying what types of managed objtects are needed, information flows between managed objects and managers and the overall structure of management information which must be stored and analysed, such as event logs, or monitored information summaries. The structural and dependency relationships between information units must be determined, as well as quality attributes such as availability requirements, atomicity and persistence.

4.3 Computational Viewpoint

A managed object in this viewpoint is likely to be a software entity or data object representing a hardware (real-world) entity. For example a resource scheduler has a data structure representing a computer, which it manages for work allocation. Assuming the computer also acts as a gateway interconnecting two networks, there will be a gateway object which is a set of routing tables managed by a routing manager. This example illustrates there can be many different types of objects being managed for different purposes which represent the same real-world entity.

The emphasis in this viewpoint is on the definition of interfaces to objects in terms of a detailed specification of interactions which can be performed on it and visible state information. Automating management and specifying management alagorithms is part of this viewpoint.

Computation domains are likely to correspond to services and the functional partitioning within those services.

The structuring of management in this viewpoint is independent of the computer systems and networks on which it is implemented and the computational viewpoint managers and managed objects are independent of implementation issues such as whether they are centralised or distributed.

In the following sections we identify the computational model for managed objects being developed as part of the Domino Project whose objective is to define a framework for Domain management in Open Systems.

4.3.1 Computational Managed Objects

A managed object has a well defined (abstract) interface which specifies interaction points and attributes. An interaction point corresponds to an operation which can be invoked on the object or a message it generates. Each interaction point in an interface has a unique identifier and defines:

Interaction Protocol Type - i.e. whether asynchronous or synchronous, unidirectional or bidirectional, error characteristics assumed from the communications system etc. In practice we assume a small set of predefined protocols such as asynchronous messages and remote procedure calls (RPCs) will be offered by an object support system.

Interaction Signatures - the data types of parameters in messages or procedure calls.

The *attributes* are internal variables or constants visible at the object's interface. This can be considered "syntactic sugaring" to avoid the need for specifying a read or write operation for externally accessible data. Typical attributes include object identifier, template identifier from which the object is created, and application specific information such as error statistics, performance information or maximum number of buffers. An attribute may be a simple data type (e.g. integer or real) or structured data type such as a record, array or set. Note that some attributes may be read-only from a manager's viewpoint, that is they are changed only by the object itself, as a result of changes in the internal state of the object. Others, such as object identifiers are constants. Attributes are analogous to set points and state variables found in control systems.

In ODP terminology an interaction point corresponds to an action and they define an interaction point to be a *set* of points in space and time at which an interaction can occur! [ISO SC21/7 89a]. ODP do not have the concept of an attribute. OSI managed objects have management operations which correspond to the above interaction points and have attributes similar to those discussed above.

A *template* is an implementation specification (e.g. in a prgramming language) from which instances may be created. The template may implement multiple abstract interfaces and a particular abstract interface specification may be implemented in different templates. Thus there may be different implementations of an interface specification. A managed object will have at least one management interface, but will also have others which support the normal functionality of the object. The template defines the internal representation of attributes as well as the code for implementing interactions, and may also specify default values for attributes.

Our template corresponds to an ODP class template, whereas OSI use the term template to mean a notation for specifying managed objects. In fact an OSI managed object corresponds to our abstract interface. They do not have the concept of an implementation of an abstract interface [ISO SC21/4 90]

4.3.2 Operations on Managed Objects

In this section we suggest a generic set of management interactions which are needed for the management of objects. In addition to the basic interactions there will be others which are dependent on the particular type of object e.g. for configuration or fault management. However we do not consider these in this paper. Some interactions with an object may affect the real world entity which it references, whereas others only affect the managed object itself.

Read Attribute - this permits a manager to obtain information on the current state of an object in order to make management decisions.

Write Attribute - this permits a manager to modify the performance of an object for example by changing a set-point or error limit. There will be application specific constraints on what values can be written to an attribute. The input parameters specify the attribute name and corresponding values to be written.

Application Specific Operations - these are operations which may alter an object's state, trigger off further interactions with other objects, and then return a response to the initiator. An operation may reference multiple attributes. For example a boolean expression involving some attributes may be used as a guard on changing the values of others.

Notifications - these interactions are operations by the managed object on the manager, in the form of unsolicited reports, for example notifying error conditions or regular reports on the state of the object.

Read and *write* an attribute are base default operations which apply to a single attribute. More complex interactions involving arbitrary groups of attributes which are to be read or written can be defined as application operations.

In some applications the manager may itself be a managed object which implements the same management interface as the objects it is managing. The manager may then perform management operations on itself. An examples is a configuration management process which performs interface binding for other processes and also binds its own interface.

4.3.3 Operations on a Domain

Creating and destroying objects such as software components which are part of a distributed application can be rather complex. Creating could involve loading code and may invlove the interactin of factory objects within the domain. Destroying may require the object to interact with its peers to reach a consistent state before being destroyed. Objects are always created in a domain in our model so these are considered as domain operations in section. Domains are themselves managed objects which can be created or destroyed.

Create object - this creates an object from a specified template and then sets its attributes according to parameters specified or according to default values.

Destroy object - this removes the object from the system and recovers any resources allocated. There may be constraints in that an object can only be destroyed when in a passive or stopped state.

Other operations which can be carried out directly on a domain include:

Include Object in a Domain - This includes an object into a domain by adding its identity to the object set of the domain. It does not affect the state of the object or its membership of other domains. The effect is to make an existing object a member of an additional domain without creating a new instance of the object.

Domain objects can themselves be included in other domains. This enables powerful structures of subdomains to be created. The fact that a subdomain may be a member of more than one domain results in potential cycles of domain membership. This can be very difficult to prevent as domains may be distributed. Managers which trace down the domain structure must implement loop detection.

Remove Object from a Domain - This removes an object from a domain without affecting the state of the object. Removal of an object from a domain is carried out by removing its identity from the object set. It is only permitted if it is still a member of another domain's object set - otherwise the appropriate operation is to destroy it. This is to ensure that objects always remain within the domain hierarchy.

List objects in a Domain - enables managers to determine the members of the object set of a domain.

An operation to *move* objects between domains, including them in the destination domain and removing them from the source domain, can be constructed as a compound of include and remove.

4.4 Engineering Viewpoint

Managed objects in the engineering viewpoint will include processes, hardware components such as workstations, gateways or communications interfaces. The decisions in this viewpoint will be on how to implement managed objects and managers, to meet transparency or quality of service requirements and how to represent the management information - whether to use object oriented programming languages, databases etc.

Engineering viewpoint domains are likely to be structured to reflect the physical structure and connectivity of the ODP system e.g. a domain in this viewpoint would correspond to a Local or Wide Area Network.

Mechanisms are needed to support the main management functions of configuration, QOS, accounting and monitoring. Tools and mechanisms are needed to support structuring management into domains.

4.4.1 Implementing Managers

The following is the approach to structuring managers in the Domino project and is similar to that of the DEC Enterprise Network Architecture [Strutt 1989]. A manager can be constructed from four different types of entities as shown in fig. 4.1.

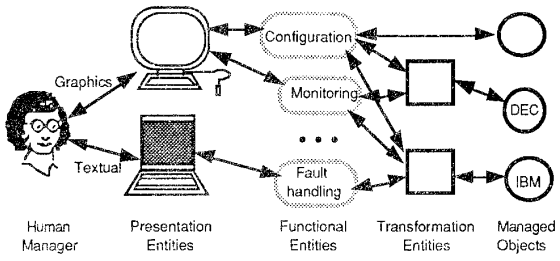

Fig. 4.1 Example Manager Structure

Human Manager - usually specifies policy but may implement the decision making for simple functional managers. A human manager is not needed if management is automated.

Presentation - this provides a human interface to management and typically could include graphical, textual for a dumb terminal, or even buttons and lights.

Functional - this implements the management functional operations. These may be partitioned according to the main management functions: configuration, security, monitoring, accounting etc. Automated management would typically be implemented in a functional manager object.

Transformation - this permits the transformation of an interface provided by a managed object into one expected by a manager. For example there will be transformations for managing OSI compatible components, for MAP MMS components, and for proprietary components. The transformation may not necessarily be between different protocols. For example a manager agent may act as an interface to a set of objects and transform a single interaction into an interaction with each member of the set (c.f. OSI management agent in Fig.3.2).

Failure of any one of the above components should not affect the normal functionality of managed objects but may prevent them being managed. There may be multiple instances of any of the manager entities and they can be physically distributed.

4.4.2 Implementing Domains

Every object has a globally unique *identifier* to enable the system to refer to it, however the identifier does not carry any meaning, i.e. it does not indicate location or which domain the object is in. Identifiers do not change when an object is moved from one domain to another. There are a number of suitable mechanisms for creating unique identifiers. For example, identifiers could be formed by concatenating the unique internet address of the computer on which an object is created with a timestamp.

Objects may optionally have a *name* by which they are known within a domain. These names need only be unique within the domain. These are really aliases which can be considered part of the human interface to management to permit human managers to refer to objects by meaningful names. They are not needed by automated managers.

We assume the availability of a location server to perform object identifier to address mapping. Domains could also cache the addresses of the objects within the domain.

OSI managed objects are assigned unique names using a containment hierarchy which reflects their physical location. For example a communication layer object will be contained in an endsystem object. This will not work for objects which can migrate from one computer to another.

Objects would have to be informed of the domains in which they are direct members. This makes it easy to an an object for its parent domains.

We are currently experimenting with domain implementations for configuration management. One approach is to have domain servers which hold object names, identifiers and addresses as well as access control lists to implement access rules. This information is structured according to domains. Some domains would have to be replicated for reliability i.e. a domain could be held on more than one server.

In this approach the main functionality of management is not implemented by the domain objects but by the manager objects. Domains are kept very simple. There can be multiple managers and they can perform management operations directly on objects once they have identified the objects in the domain. Managers need not directly address objects or even know what objects are in the domain if some form of group addressing is supported by the communication system.

4.4.3 Implementing Constraint Evaluation

Implementing access rules in terms of capabilities or access control lists are explained elsewhere [Moffett 90, Twidle 88], and so will not be covered here. However there are a number of issues with respect to evaluating internal constraints.

A membership constraint can be based on an arbitrary attribute of an object. What happens if an operation on an object changes the value of the attribute such that the constraint is violated and so the object should no longer be a member of the domain? If the object is automatically removed from the domain, then it could result in it not being a member of any domain and hence violating the principle that a managed object must always be a member of at least one domain. It is quite difficult and time consuming for a domain object to automatically check constraints for objects which are distributed around the network. An alternative is to implement the constraint checking within the object and refuse operations which would violate a constraint. This implies the constraints are coded into the object template or the object can interpret a set of constraint specifications given to it when it becomes a member of a domain. The former results in loss of flexibility in that it becomes difficult to move objects between domains and the latter complicates simple objects.

The solution adopted was to consider constraints as preconditions to domain operations. There is not automatic dynamic checking of subsequent constraint violation. A specific application can provide a manager which performs background evaluation of constraints and can take an application dependent action when it discovers an object which violates a domain constraint.

4.5 Technology Viewpoint

This viewpoint includes the technical artifacts (realised components) from which distributed systems are built. For example the particular hardware and software available for operating systems, input/output devices, storage, communications interfaces etc.

Our current implementation is based on support for configuration management using the Conic Toolkit for building distributed systems [Magee 89]. However this only supports a single programming language. A more flexible set of tools is

being developed as part of the Rex project [Kramer 90b]. These will support heterogeneous programming languages and permit abstract interfaces to be specified independent from templates which implement objects.

5 DISCUSSION

Although it is possible to use the ODP viewpoints as a means of structuring the discussion on management, this ends up with rather an artificial structuring which does not help with the understanding or specification of management in distributed systems.

The first criticism of the viewpoints is that they are easily confused with the stages in the lifescycle model for design of systems namely requirements specification, system analysis, component specificaton, component implementation and mapping onto hardware. However this is an incorrect interpretation of the viewpoint concepts which are meant to be a complementary set of abstractions of the original system in order to study different operational characteristics.

We do not disagree that there should be multiple viewpoints but we think there are an indefinite number of viewpoints rather than 5. For example the ODP viewpoints do not include a configuration viewpoint which is a structural view of the system in terms of object instances and their interconnections. This should be separated from the algorithmic view of the functional behaviour of a particular object [Magee 89]. We using this configuration view as the common framework for a set of tools for specification, design, construction, analysis and evolution [Kramer 90, Kramer 90b, Magee 90] in the Rex Esprit project. Within a particular level of abstraction there may be multiple viewpoints corresponding to different manager or user roles. Thus the modelling framework should support mappings and transformations between any number of viewpoints.

Although we have considerable experience in providing tools for building distributed sytems and using them for teaching and research, we have had difficulty in applying ODP definitions and terminology [ISO SC 21/7 89] to our distributed processing concepts.

A problem with both OSI management and ODP is the emphasis on inheritance for object specification, and on maintaining class hierarchies as a means of determining object compatibility. Inheritance is not practical in distributed systems if classes in the inheritance hierarchy are remote [Snyder 86, Black87]. Maintaining a strict hierarchy of classes when the classes are being specified and implemented at different sites in multiple organisations around the world will also prove impractical in our opinion as it would require a global database of classes.

In our opinion object compatibility has to be based on comparing interface specifications using structural type checking or a type conformance rules [Black 87]. In addition behavioural specification should be checked for compatibility, but that is still a subject requiring further study.

6 ACKNOWLEDGEMENTS

I wish to acknowledge the contribution of Jonathan Moffett who helped to define managed objects and domains. Shing Chi Cheung, Naranker Dulay, Jeff Kramer, Jeff Magee and Kevin Twidle have also contributed to the concepts described in this paper. The work was funded by DTI/SERC under grant number GR/F 35197.

7 REFERENCES

[Black 87] Black A, Hutchinson N, Jul E, Levy H, Carter L. Distribution and Abstract Types in Emerald, IEEE Trans. on Software Eng., vol. SE-13:1, Jan. 1987, pp. 65-76.

[ISO SC21/4 89] ISO/IEC JTC1/SC21 Proposed DP 10040 OSI Information Processing Systems, Systems Management Overview, Draft, Dec. 1989.

[ISO SC21/4 90] ISO/JTC1/SC21 Proposed DP 10165-1 OSI Management Information Services SMI Part 1, Management Information Model, Feb 1990.

[ISO SC21/7 89a] ISO/IEC JTC1/SC21 N4025 Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model, (Working Document), 1989.

[ISO SC21/7 89b] ISO/IEC JTC1/SC21 N4022 Basic Reference Working Document on Topic 4.1 - Structures and Functions, Nov. 1989.

[Klerer 88] Klerer S.M.,The OSI Management Architecture: an Overview, IEEE Network, Vol. 2:2 (March 1988), pp 20-29

[Kramer 90a] Kramer J, Configuration Programming - a Framework for Development of Distributable Systems, Proc. IEEE Int. conf. on Computer Systems and Software Eng. (CompEuro 90) Israel, May 1990.pp.

[Kramer 90b] Kramer J, Magee J, Finkelstein A, A Constructive Approach to the Design of Distributed Systems, 10th Int. Conf. on Distributed Computing Systems, Paris May 1990.

[Magee 89] Magee J., Kramer J. & Sloman M.S., Constructing Distributed Systems in Conic, IEEE Trans. on Software Eng., Vol SE-15:6, June 1989, pp 663-675

[Magee 90b] Magee J, Kramer J, Sloman M, Dulay N, The REX Software Architecture, Distributed Computer Systems in the 1990s, Cairo, Sep. 1990.

[Moffett 90] Moffett J, Sloman M, Twidle K, Specifying Discretionary Access Control Policy for Distributed Systems, Domino A1/IC/3, March 1990.

[Sloman 88] Sloman M. (ed), Distributed System Management, in Issues in LAN Management, ed I. Dallas & E. Spratt, pp 15-46, North Holland 1988.

[Sloman 89] Sloman M. S. and Moffett J.D. Domain Management for Distributed Systems, in Meandzija & Westcott (eds), Proc. of the IFIP Symp. on Integrated Network Management, Boston, USA, May 1989, North Holland, pp 505-516.

[Snyder 86] Snyder A., Encapsulation and Inheritance in Object-Oriented Programming Languages, OOPSLA '86 Proc., pp.38-45.

[Strutt 89] Strutt C, Shurtleff D., Architecture for Integrated Extensible Management Director, in Meandzija & Westcott (eds), Proceedings of the IFIP Symposium on Integrated Network Management, Boston, USA, May 1989 (North Holland 1989), pp 61-674.

[Twidle 88] Twidle K, Sloman M., Domain Based Configuration and Name Management for Distributed Systems, Distributed Computer Systems in the 1990s Workshop, Hong Kong, Sept 1988, IEEE Computer Society, pp.147-153.