

Middleware and Management

Concepts, Characteristics, and Integration

Dr. Sven van der Meer
Telecommunication Software and Systems Group
Waterford Institute of Technology

Version 1, 28/04/2003

1 Integration of Management and Middleware

Do current management frameworks offer answers to the new questions? Are they applicable for future applications, services, and resources? Answers to those questions can be given after analyzing current trends in communication and computing and after evaluating the actual level of interworking and integration of management and middleware. Doing this, the answer is *no*, under most favorable conditions *maybe*.

The current situation can be described by the interworking (*not* the integration) of middleware and management systems. Furthermore, the available information sources within companies and organizations rely on many different solutions for storing data. They are still waiting to be harmonized. In short, this situation can be described as follows.

1. **Distributed applications can access classic management systems** via ‘gateways’ that translate specifications and interactions. Figure 1-1 shows an example where clients from the WWW¹ manage resources via CORBA². The figure shows the CORBA gateway to the management systems. [CORBA-MAN]
2. **Legacy management systems are used to manage distributed applications.** Providers and operators use their running management systems to administrate distributed applications. Thus, the investment for the education of this staff is not wasted but reused for distributed applications. [CORBA-TMN]
3. **A few middleware concepts and many different products are used in parallel.** Most of them provide mechanisms for interworking (that means for the invocation of operations and the exchange of information). The information exchange often needs an informal agreement of developers on semantic. Each middleware concept includes basic management functionality on object level (e.g. based on the clustering concepts of ODP³; [Raymond95]). Furthermore, “middleware developers strive to support applications that meet the technical challenges of ubiquitous computing” [Geihs01].
4. **Service platforms form a layer of abstraction** enabling service creation and deployment, monitoring of distributed applications, and integration of legacy systems. Within those platforms, the management of services is included by means of the support of tools for the definition of services and business roles, an execution environment for services, and the management of the platform itself. [Funabashi00] [Magedanz01]

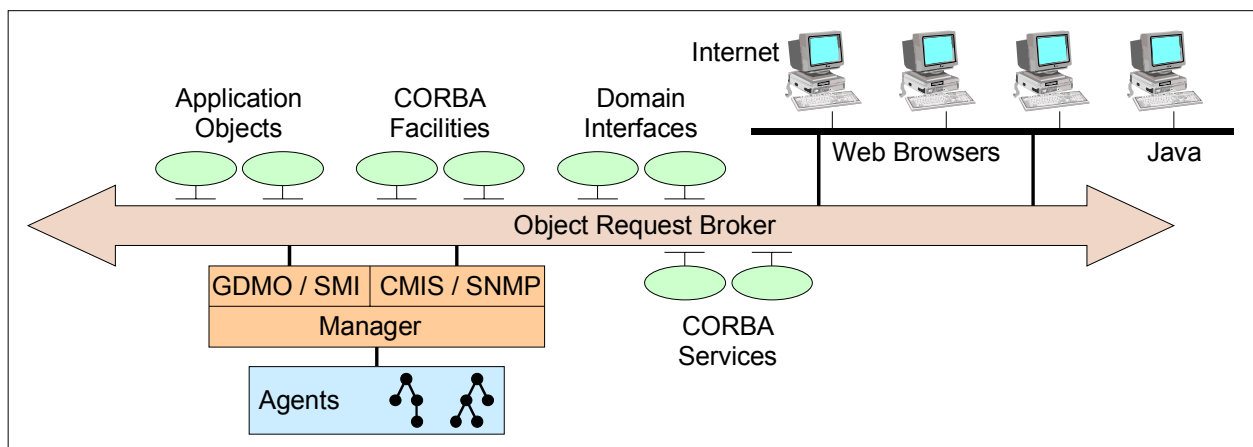


Figure 1-1: Distributed Applications accessing classic Management Systems [NMF-GB909]

¹ World Wide Web

² Common Object Request Broker Architecture

³ Open Distributed Processing

Management is defined as all activities needed to operate communication networks and services in a secure and effective way. It identifies methods and provides tools to support configuration, monitoring, maintenance, and administration of these networks and their services. The target vision is to support user and provider in planning and operating networks and distributed systems. Nevertheless, middleware and service platforms do not reflect management standards sufficiently. Each of them comes with a new concept for managing objects or services. Furthermore, development of middleware does not follow management principles. The components object specifications, protocols, and data formats are designed specifically to support distribution. Here, an integration of management concepts into these components can be an important step. [Hegering99]

Distributed applications define new requirements, which middleware and management need to address. Applications and devices are not realized with a single technology. Devices and network appliances need to be accessed in a technology independent manner. Specifications must be independent of dedicated vendors. At the same time, the support of autonomy of operators and vendors is needed to enable them to offer clearly distinguishable products. These are basics for cooperative environments in a competitive world.

2 Characteristics of Middleware

The term middleware is used today to describe any technology that provides an abstraction from the heterogeneous components of computer environments, like hardware platforms, networks, protocols, operating systems, applications, groupware, and databases. Middleware ties components of a distributed application together and solves the problem of scalability. It supports application deployment using well defined, probably standardized, Application Programming Interfaces (API). Those APIs and related interfaces decouple the applications from technologies as shown on Figure 2-1.

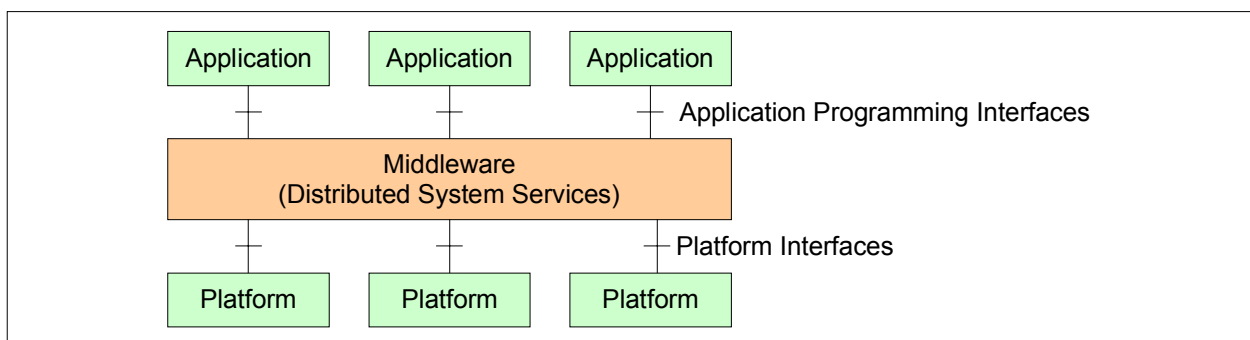


Figure 2-1: Middleware – Logical View of Middleware

Two different types of middleware are present: Inter Process Communication (IPC) and Distributed Transaction Processing (DTP). IPC based middleware systems operate with a direct communication among distributed components. Members of IPC based middleware are Remote Procedure Call (RPC) middleware, Message Oriented Middleware (MOM), and Object Request Broker (ORB).

DTP based middleware realizes the link between a client and any kind of database. Those middleware systems define abstract interfaces to databases for handling transactions and concurrency. Members of DTP based middleware are portable Transaction Processing (TP) Monitors and interconnected Database Management Servers (DBMS).

2.1. Paradigm of Client – Server

Distributed object communicate in a client/server relationship. The client calls operations on a server and receives return values. The server offers those operations on its interface(s), runs the business logic once an operation is called, and generates the return value. The technical realization of interfaces and the communication is task of a middleware.

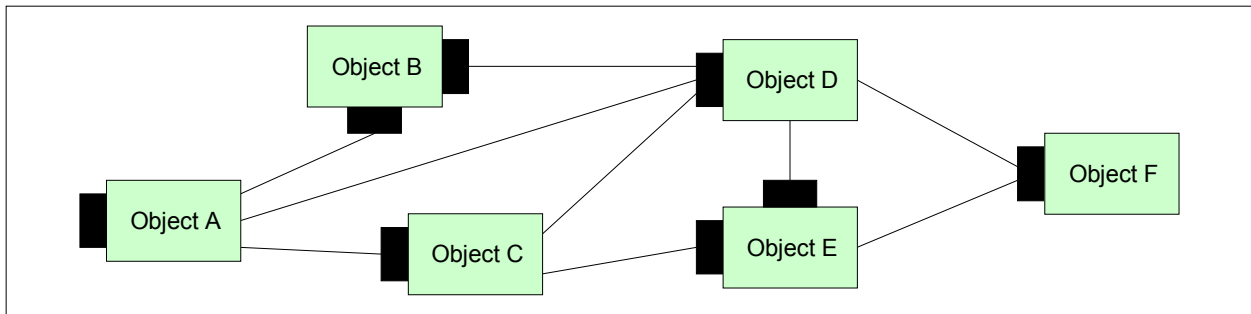


Figure 2-2: Middleware – Distributed Objects

The actual relationships between distributed objects are not defined by the paradigm. Each object can access each other object. The definition of the relationships needs to be done in the development process of the distributed application. Hierarchical structures are not supported. They need to be realized individually.

2.2. Message Oriented Middleware

Message Oriented Middleware accomplishes the communication of application components through the exchange of records called messages. Messages are strings of bytes that have meaning to the applications that exchange them. MOM is event-driven. It can support asynchronous as well as synchronous communication. However, there is no general standard or common set of specifications. At least three different techniques for the exchange of messages are available.

Message passing applies a direct communication model. A message is sent directly from one application to another. The model is connection-oriented. A logical connection between applications needs to be established. The exchange of messages can be either asynchronous (via a polling model or by call back routines) or synchronous (sender blocks until a message is returned).

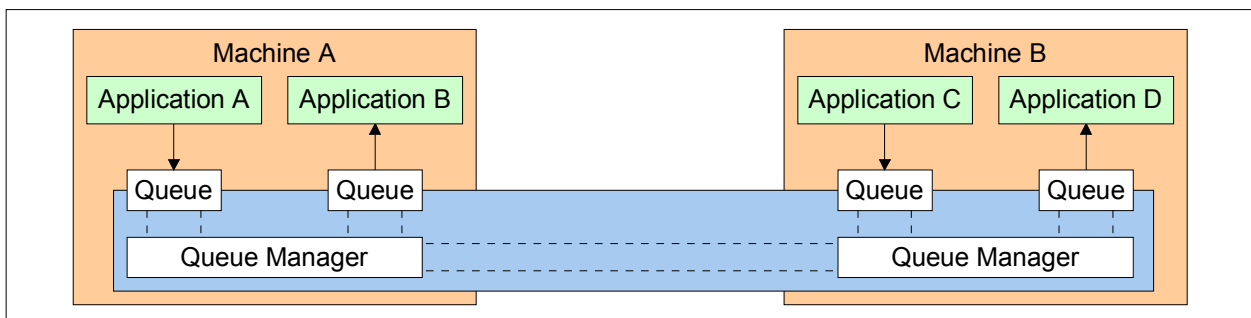


Figure 2-3: Middleware – Message Queuing

Message Queuing applies an indirect communication model. Applications communicate via a message queue. This model is connectionless. Messages are put in queues for immediate or subsequent delivery. A Queue Manager is responsible for message handling and delivery. Message queuing implies the support for different types of Quality of Service (QoS), such as reliable message delivery (no packet loss on the network), guaranteed message delivery (messages are delivered immediately or eventually, at least after a specified period of time), and assured non-duplicate message delivery (message are delivered only once).

Publish and Subscribe is based on trading. Publishers produce information and offer them. Subscribers sign in to certain topics and receive messages. Publishers and Subscribers usually do not know the existence of the other. Applications can be loosely coupled increasing the flexibility of the design and operation of a system. Those systems can be reconfigured dynamically without interrupting operations. New applications can be added without disturbing existing applications.

2.3. Remote Procedure Call

The Remote Procedure Call is a common accepted and widely used technique for the communication between applications in distributed environments. All major operating systems support client/server communication via logical RPC interfaces. That is, clients and servers have local, logical interfaces that are called stubs. Those stubs realize proxy objects for remote functions. To the client, the stub looks like the remote procedures thus providing location transparency (hiding the actual location of the server).

The logical function calls on the RPC interface are mapped to physical invocations on the stub. The later one has the knowledge on how to access the network's transport layer to send the request to the server and to receive the reply for the client. The client itself usually blocks until the server replies to the remote call. On server side, the stub calls the requested function (procedure) as if the client would have done if called locally. The programmer of the server has to provide some functionality to support a remote call, like directory, security, etc.

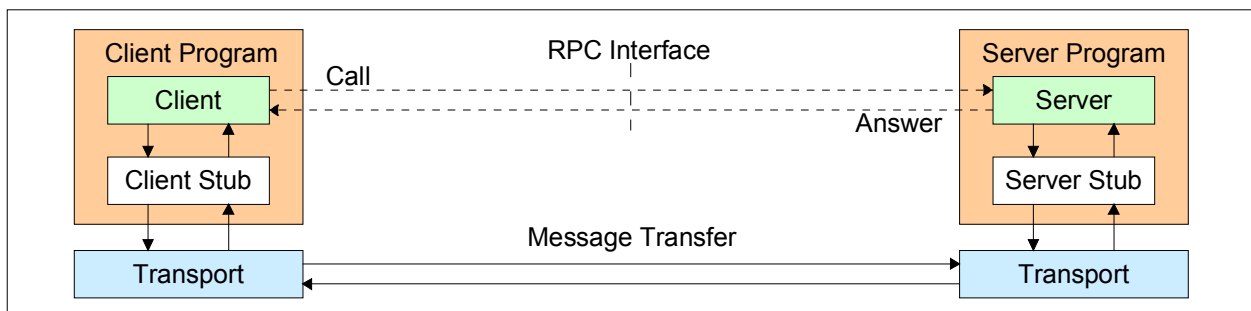


Figure 2-4: Middleware – Remote Procedure Call

To use the network's transport layer for data exchange between client and server, the data has to be streamed through a communication channel. The serialization of the data for the transport inside of the network is called marshalling. The serialization requires that all data to be transported be pre-described, including type, format, and length. Such a description is to be provided by the programmer of the server in form of an interface definition using an Interface Definition Language (IDL). An IDL is a high level, universal notation that is capable to describe interfaces independent of the actual programming language.

2.4. Object Request Broker

Object Request Brokers can be classified into a group compliant to the Object Management Group's CORBA⁴ and one compliant to Microsoft DCOM⁵. In the following, the generic ORB architecture is viewed that is the basis for both.

The ORB is a software component that enables objects to initiate requests and to receive responses. All inter-object communication happens via the ORB, independent of the actual location of the objects (local or remote). The ORB enables objects to communicate over networks with different communication protocols and to reside on different hardware platforms. It provides the mechanisms by which objects make requests and receive responses. Furthermore, it is responsible for managing and locating the objects, supporting both inter-object communication and communication between objects and external services.

Most ORB products provide a set of components that is specified by the Object Management Group (OMG). At first, an IDL is used as notational language to describe interfaces of objects. It is implementation-neutral and needs specifications for the mapping to implementation languages. An Interface Repository contains all IDL definitions for attributes, operations, user-defined types, and exceptions. The Basic Object Adapter (BOA) represents the interface between the ORB and server applications. It dispatches objects that the server application maintains, and exchanges messages with the server objects. The Static

⁴ Common Object Request Broker Architecture

⁵ Distributed Component Object Model

Invocation Interface (SII) is a stub-based interface used by client programs in order to invoke services on application objects. Finally, the Dynamic Invocation Interface (DII) provides a generic interface that does not require stubs, but supports dynamic construction of object invocations by the client program at run-time.

The ORB establishes a client/server relationship among objects. A client can transparently invoke any method on any server object. The ORB intercepts the call, seeks for an object that has implemented the call, passes the parameters to that object, invokes the method, and returns the result. Clients can invoke either 'one way requests' when no result is expected or synchronous requests.

The ORB itself is responsible to locate the server object by use of the implementation repository, to exchange data between client and server (including marshalling), to invoke the server (dynamic server invocation), and to recover from failures of the server object. The ORB provides objects with services like naming, lifecycle, property, relationship, query, and licensing.

2.5. Distributed Transaction Processing

Transaction Processing has been used for mainframe-based applications where TP Monitors have managed the limited number of connections to databases. They acted as connection concentrator reducing overhead and increasing performance of database access. Today, TP Monitors are the only solution for transactional integrity in database environments. They support the so-called ACID properties that describe the quality of transactions:

- **Atomicity:** All operations that an application performs, which involve updates to any kind of resource, are grouped into a "unit of work." This unit is referred to as atomic, meaning it is indivisible. Any partial completion (due to system failures) will be rolled back.
- **Consistency:** At the end of a transaction, all resources that have participated will be in a consistent state.
- **Isolation:** Concurrent access to shared resources by different units of work (performed by different applications) is coordinated so that they do not affect each other. Transactions that compete for resources are isolated from each other.
- **Durability:** All updates to resources that have been performed within the scope of a transaction will be persistent or durable.

The standard for DTP represents a basis for TP Monitor products. It allows programmers to share resources keeping the overall system in a consistent state. Databases are connected to the TP Monitor via a standardized interface called XA. Applications use the TX interface to communicate with the TP Monitor. The Standard Transaction Definition Language (STDL) provides a TP Monitor independent API in form of a higher layer, procedural language for the description of transactions.

3 Characteristics of Management

Management is needed to control and to optimise the operation of a network and offered services, and to respond to changing user requirements. Management contains all activities for the secure and effective operation of networks and services. Those activities are methods and tools for the initialisation, monitoring, maintenance, and administration of network functions, services, and application components. Management activities can be performed explicitly by human operators or automated by dedicated software modules. Management systems support different independent actors that want to co-operate (support of autonomy and co-operation), large amount of involved components (scalability), huge variety of employed systems and components (vendor independence, standard based management), and frequently changing systems and components (reusability of management concepts, support of the whole lifecycle of network equipment and services).

Several approaches are in use for the development of management systems responsible for complex management activities:

- **Top-Down Approach:** The analysis of the requirements the system to be managed leads to specifications of business roles (data), functional processes and services (function), and network locations where the business operates (communication).
- **Partitioning:** The task of managing a complex system is subdivided in to several domains for the delegation of management responsibilities. Domains can be service oriented (signalling network, packet switched network), functional areas (FCAPS⁶), layers from the RM-OSI⁷ (vertical management of (N)-Layer), layers from TMN⁸ (vertical management of business, services, networks, and network elements), geographical (horizontal), and business domains (horizontal).
- **Object-oriented:** Distributed applications and management components are as dissociated as possible (almost no overlap in functionality) with simple interfaces (making them reusable). Information is exchanged in form of standardized messages in an abstract form that is independent of the internal structure of the object.
- **Views:** Management systems can be specified for example following the guidelines of Open Distributed Processing (ODP) or the Telecommunication Information Networking Architecture (TINA) and its Management Architecture.

Each approach can be used separately or in combination with other approaches to define a reference model for a management system. Such a reference model includes definitions for the management information, management services, and management applications. Care has to be taken in order to optimise the depth of partitioning and decompositions. If that process remains too abstract, the reference model may become useless. If it becomes too detailed, the reference model might not be universal and extensible enough, and may not have the ability to evolve as the managed systems evolve.

3.1. Management Architectures

Management architectures have the task to provide a principle order of functionality and information specifications for a management system. They enable a high level abstraction for designers of management applications and provide guidance for the design of management protocols and services. They consist of concepts (like simplicity or openness), rules specifying how to use the concepts, and models visualizing how an application of the concepts and rules can lead to design specific classes of management systems.

Classical management architectures are the Simple Network Management Protocol (SNMP), the OSI Management (X.700), and the TMN. Other developments have spent huge effort to reuse concepts and rules from those management architectures for a specific application domain. The Object Management Architecture (OMA) responses to the requirements object-oriented distributed applications. TINA specifies telecommunication service platforms. Mobile Agent based management realizes management by delegation with mobile code segments. Directory-enabled Networks (DEN) harmonise classified databases. Web-based Enterprise Management (WBEM) aims to manage business processes and value chains on top of distributed objects and management systems.

3.2. Paradigm of Managers and Agents

Management architectures are based on the well-accepted paradigm that management information is exchanged among **Managers** and **Agents**, as shown in Figure 3-1. **Managers** are software systems, responsible for the communication with agents to retrieve information about their state and to change the state, the storage of the obtained information and own activities in adequate databases, and the provision of the stored management information to administrators (through user interfaces) or to senior management systems. An **Agent** is a software module of a certain network component (e.g. a bridge / router) entrusted

⁶ Fault, Configuration, Accounting, Performance, and Security

⁷ Reference Model for Open System Interconnection

⁸ Telecommunication Management Network

with the supervision, configuration, and control of the entities of resources and the connection with the related manager and the transmission of the requested information to the manager.

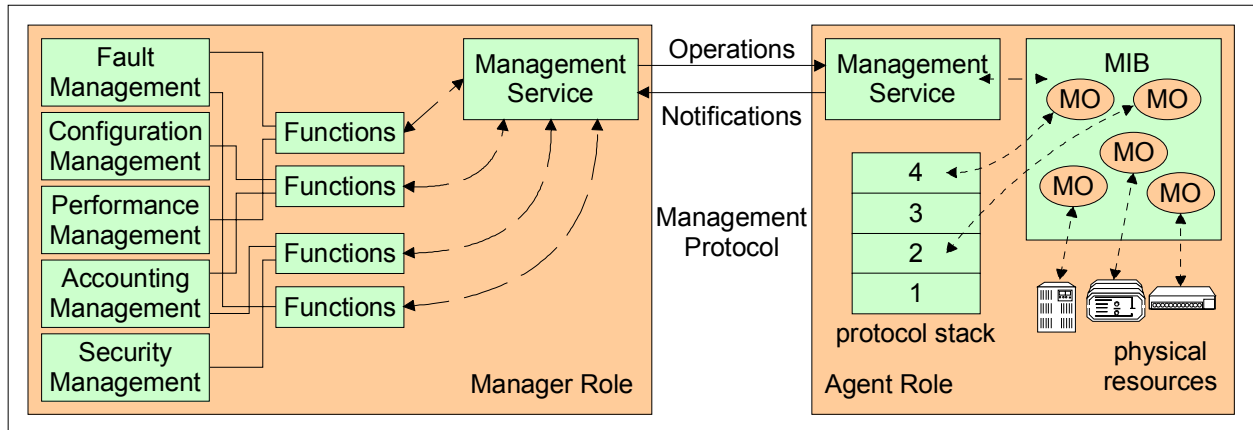


Figure 3-1: Management – Manager and Agent Role [Badach94]

Network Components are modelled as managed objects. They represent an abstraction of physical (networks, interconnection equipment, customer premises equipment) or logical (protocol stacks, switching/routing tables, status information) resources. Managed Objects (MO) can be accessed via a virtual information store, termed the Management Information Base (MIB). Objects in the MIB are specified using abstract notations. Each object type is identified by a unique and administratively assigned name. A set of objects forms a module in the MIB that provides access to management information for a clearly defined domain (e.g. monitoring or accounting).

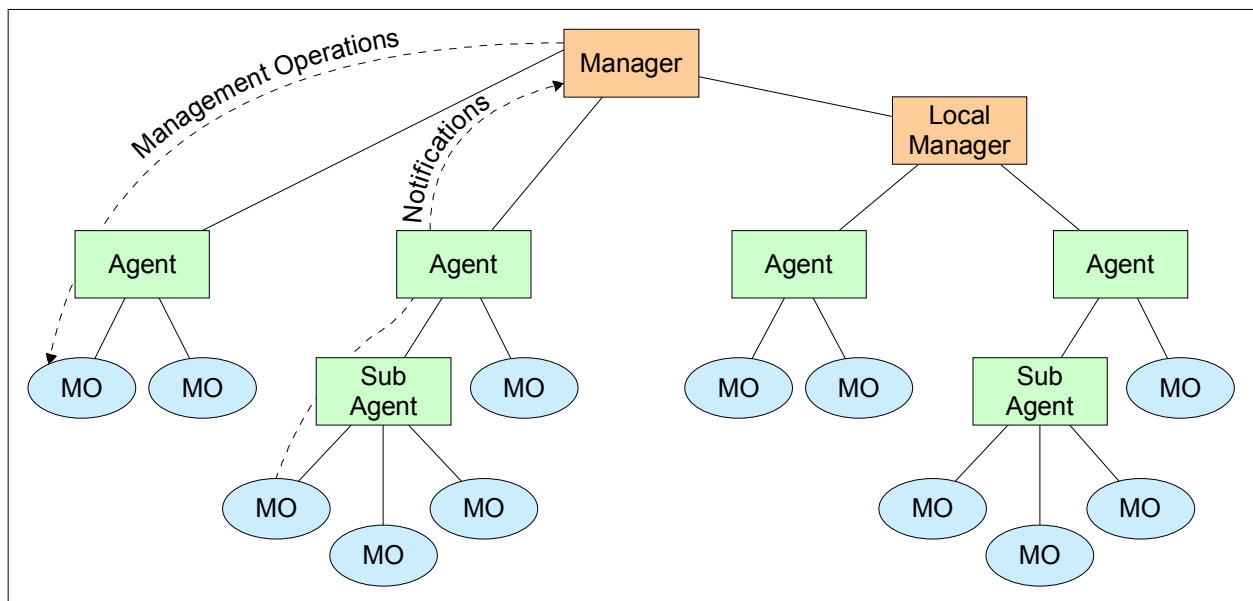


Figure 3-2: Management – Management Hierarchy

Managers and agents exchange information in form of operations and notifications. In combination with managed objects, they build hierarchies. This hierarchical management model is shown in Figure 3-2. It describes a hierarchy of entities in the role of managers, local managers, agents, sub agents, and managed objects. Each managed object is assigned to exactly one agent. In distributed environments, a managed object might be assigned to multiple agents, too. A number of agents can be arranged in the hierarchy among managed objects and managers. Local managers and sub agents are introduced to allow the formation of domains (organizational, geographical).

A manager needs to be provided with the knowledge of each entity of the underlying hierarchy, their configuration functions, and their relationships to each other. Managers offer management operations that combine multiple configuration functions of subordinate managers, agents, and managed objects to units of work (also called transactions). Graphical User Interfaces (GUI) or superior management systems can use these management operations to perform complex management tasks in a simple way.

An agent is responsible for the controlling of assigned sub-agents and managed objects. This includes the evaluation (e.g. authorization checks) of management operations and their forwarding to the configuration functions of managed objects, as well as the processing of notifications and their transfer to addressed managers.

3.3. Management Models

The specified concepts and rules of a management architecture form architectural models. Management architectures differ in the terminology for their models. However, management architectures contain at least definitions that can be grouped into four models: Information Model, Functional Model, Communication Model, and Organizational Model. All those models are related to a specific part of the general Manager-Agent paradigm, as shown in Figure 3-3.

3.3.1. Information Model

The Information Model provides notational techniques for management information, specifications of managed objects, and specification of MIBs. Hereby, an object is a collection of behaviours that share the same state. A state describes a stable internal condition of an object. The behaviour explains a change in the state of an object. Attributes are a set of data values and qualifiers over those data values with fixed semantics. An operation of an object is a behaviour, normally initiated by the reception of a notification. A method is the implementation of a behaviour and parameters are data values with the syntax conform to a prescribed protocol. An event is an autonomous behaviour signalled by the emission of a notification.

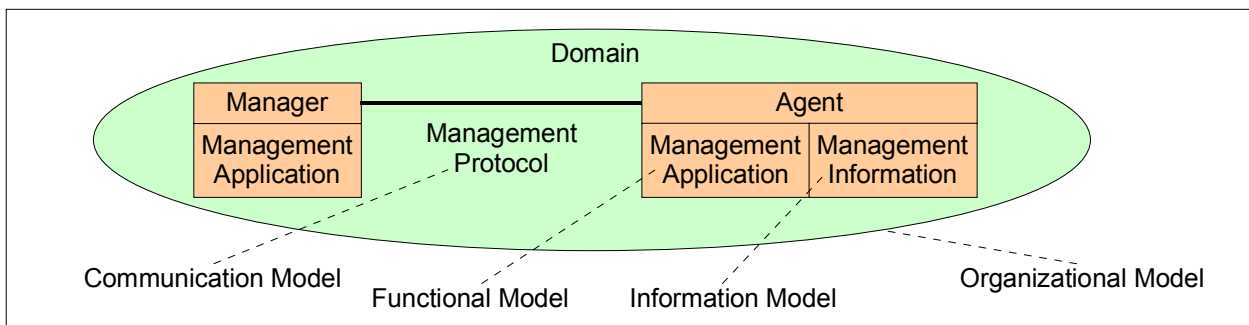


Figure 3-3: Management Models

3.3.2. Functional Model

Management is required for a number of purposes categorized into a number of functional areas: fault, accounting, configuration, performance, and security management. These areas have been recommended by OSI management. Specific management functions, within these functional areas, are provided by OSI management mechanisms. Many of the mechanisms are general. Similarly, managed objects are general in the sense that they may be common to more than one functional area.

3.3.3. Communication Model

The communication between a manager and an agent is done by the exchange of management information in form of operations and notifications. The manager requests certain behaviour via an operation and

the agent can use notifications to inform the manager about detected events. Therefore, communication between managing entities is done through status requests, the exchange of control information, and event notification. The communication model provides guidelines for the identification of communication parties, specifies mechanisms for the communication between them, and defines formats for syntax and semantic of the communication.

3.3.4. Organizational Model

The organizational model depicts concepts for the adaptation of the management architecture to the organizational structure of an enterprise. This can be related to a certain business role the enterprise is acting in (service provider, network operator), to domains (e.g. the management of groups of resources in a room, a floor, a building), or to management policies.

3.4. Character of Management Systems

Following the rules and concepts of management architectures, a real management system can be designed and implemented. The first solutions have realized a centralized approach, where one manager controls the entire network. These solutions have proven to be unable to efficiently manage even small networks and a few services. New approaches provide distribution for the configuration of managers and agents, the definition of policies to decouple management functionality that can be further distributed, handle the reuse existing resources like directories, and include emerging technologies like the World Wide Web to perform management.

Hierarchical Management means to subdivide management intelligence in a strict hierarchy of management applications. Manager requests operations on Local Managers, which forwards those requests to other Local Managers or Agents. The structure of an enterprise management system can now include the organizational requirements of a central system administration group and administrative domains for specific management functionality.

Distributed Management goes to rearrange to location of management intelligence from one central point inside of a network to different, separated computer systems or hosts, management applications, and management domains. The available resources are used in a more efficiently way and the scalability of the management, as the network or applications grow and change, guaranteed. Management applications can be loosely coupled to operate in co-operation or independently from each other.

Policy-based Management means to establish rules (policies) that the network and applications can act on in response to changing conditions. This includes the profiling of users, services, and applications to determine appropriate QoS levels.

The idea of **Directory Enabled Networks** is based on the OSI Directory [ITU-X500] and the development efforts spend on the Light-weight Directory Access Protocol (LDAP) [IETF-RFC1777]. The target vision is to unify the many kinds of directories used in enterprise networks by means of user interfaces, data formats, data duplications, data inconsistencies, and synchronization. The outcome should be a global directory as a single point of data access with a well-agreed structure. This global directory can be administered locally similar to X.500 sub trees.

Web-based Enterprise Management realizes the conception of so-called thin clients that gain access to network elements or services. Many network elements already provide interfaces for web browsers. The programming language Java, fully integrated into the World Wide Web (WWW), promises to provide the flexibility needed at the clients side.

4 Integration of Management and Middleware

The paradigm of client/server does not define the actual relationship of the involved objects. The question which object employs which interfaces is up to the design of the distributed system. It can be integrated into the business logic of the objects themselves, configured at runtime via appropriate tools, or rely on dynamic decisions of the clients.

The actual configuration of a distributed system can be determined with tools that monitor the system and reflect its global state via Graphical User Interfaces (GUI). Addressing more than one object at the same time cannot be done with built-in mechanisms of the state-of-the-art middleware platforms. Hierarchies of objects can be addressed, but are not supported by the middleware either. This situation is the reason for the multitude of different approaches for the configuration management of distributed systems that are neither standardized nor interoperable.

The realization of such a management hierarchy within distributed object environments enforces the definition of appropriate addressing schemes and interfaces. An addressing scheme must support at least the identification of agents and managed objects within the hierarchy; optionally it should also include mechanisms that allow scoping and filtering. The interfaces for managed objects, agents, and managers need to be defined in a generic way enabling their application for different middleware technologies.

4.1. Protocols and Interface Definitions

The differently paradigms of middleware and management resulted in different approaches for programming interfaces and protocols. For middleware, two mechanisms are currently in use. Client and server exchange information via messages or via a RPC. The message transfer demands a component for the queuing of messages. RPC based middleware is based on statically defined interfaces employing an IDL. The defined interfaces are processed by a compiler so that the middleware gains knowledge on the operations and the data of server objects. The communication protocols of RPC based middleware need to be provided with this information in order to guarantee the correct transmission of data. Changes in the interfaces reflect directly to the actual protocol since the compiler needs to be run again to produce the new specifications for the protocol.

Management systems are based on a clear, pre-defined operation model. Managers invoke operations on Agents and Agents send notifications to Managers. The operations are pre-defined in the management protocol. SNMP for example offers five primitives to traverse tables, and to set and to alter variables. One primitive is reserved for the exchange of notifications. The Common Management Information Protocol (CMIP) includes operations for the creation and termination of managed objects. Interfaces are defined by means of managed objects. Each managed object is specified with a set of attributes and operations for each attribute. The operations are directly inherited from the management protocol. That is, no managed object can offer operations that are not defined in the management protocol. This restriction limits the applicability of management protocols for environments other than the original management of networks and systems.

4.2. Object Models

The object models of middleware technology enable the object-oriented specification of interfaces including mechanisms like inheritance and polymorphism. Their focus lies on the signature of interface in a programming language neutral way.

From the management perspective, object models are integrated in the information model of the management architecture. An information model provides not only notational techniques for the specification of managed objects but also for the assembly of those specifications to MIB repositories. This standardization enables the interworking of management systems because all involved parties (managers) have the same knowledge of the system they are managing.

The management of the same ATM⁹ network as example can be done with management systems from different vendors as long as each management system implements business logic, which is based on the standardized ATM MIBs and as long as the nodes inside of the ATM network provide access to system information following the same MIBs. For this purpose, object models of management systems include information that describes more than just the signature of an interface. Additional information like the range of values, the behaviour of objects and operations, the intended usage of an object, policies for the access of objects, and relationships among objects can be specified.

4.3. Management on Object Level

The engineering view of ODP specifies a model for the management of engineering objects (Figure 4-1). This model is based on the mechanism for connecting engineering objects. Binding objects are called *channels* in the engineering view. *Channels* provide inter-object communication. They can be configured with infrastructure objects by means of application semantics (client and server *stubs*), binding (*binders*), and networking (*protocol adapters*).

The recommendation [ITU-X903] contributes rules for the structure of the engineering viewpoint. An engineering specification amongst the rest is expressed as a configuration of engineering objects, structured as clusters, capsules, and nodes. For those three structuring elements and for engineering objects themselves, the standard identifies management functions. The node management function controls processing, storage and communication functions within a node. The object management function is used to checkpoint and to delete objects.

The cluster management function checkpoints, recovers, migrates, deactivates or deletes clusters and is provided by each cluster manager at a cluster management interface, comprising one or more of the following functions with respect to the managed cluster. The capsule management function instantiates clusters (including recovery and reactivation), checkpoints all the clusters in a capsule, and deactivates all the clusters in a capsule and deletes capsules. It is provided by each capsule manager at a capsule management interface comprising one or more of the following functions with respect to the managed capsule.

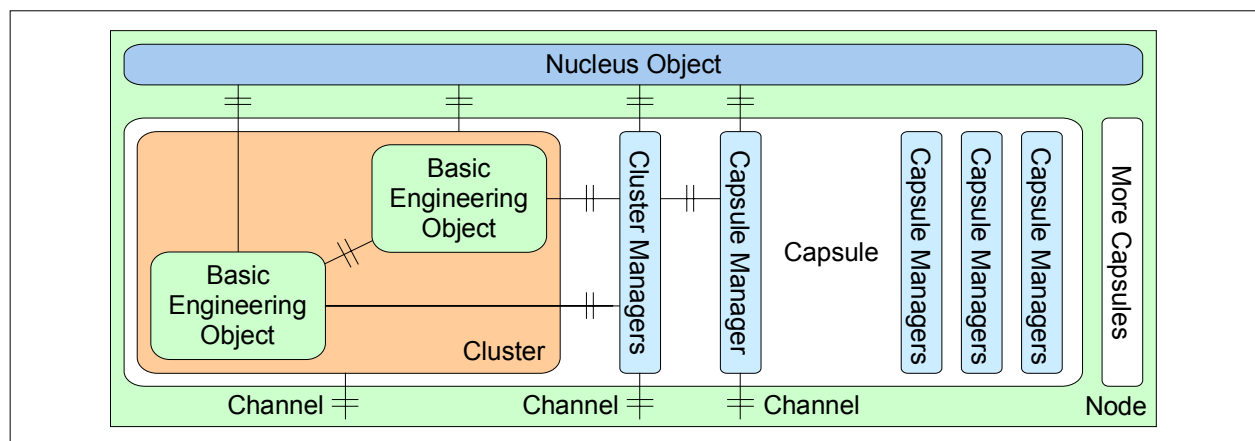


Figure 4-1: ODP Object Management

A node has a nucleus object that supports many capsules that contain many clusters that contain many basic engineering objects. All communication between clusters is realized with channels. An actual implementation of this model can impose restrictions such as only one object per cluster or only one cluster per capsule.

⁹ Asynchronous Transfer Mode

4.4. Middleware for Management

4.4.1. Common Object Request Broker Architecture

The adoption of CORBA in the management domain is considered as a step towards the integration of middleware and management as well integrating the two different approaches for management systems. SNMP and TMN employ different paradigms and use different mechanisms for information modelling, communication protocols, functional specifications, and organizational domains. The Joint Inter-Domain Management specifications (JIDM) provide one way of interworking through their capability of inter-operating with TMN and SNMP systems.

The main advantage of this middleware-management interworking is seen in the combination of object-oriented architectures with simple, flexible, and open APIs (such as CORBA and Java) with well-known and standardized protocols (such as CMIP). A manager in the CORBA domain would be able to interact with an OSI or an SNMP agent as if they were in the CORBA domain, ideally without any knowledge of the target object's domain. The resulting network management architecture is intrinsically capable of interworking with legacy CMIP/SNMP systems and ready for the provision of new information models on different specification languages. [Magedanz99a]

4.4.2. Java Management Extensions

The Java Management Extensions (JMX) [JAVA-JMX] “defines architecture, design patterns, and services for applications and network management in the Java programming language.” [JAVA-JMX] It is implemented by the Java Dynamic Management Kit (JDMK) [JAVA-JDMK]. These extensions enable Java applications to be easily managed. JMX offers a managed object server that acts as a management agent and that can be embedded in existing Java applications. Every known management solution is supported by JMX: SNMP, TMN, and web-based management. The specifications include an SNMP manager API, web-based management client, and a TMN manager. Other Java products are leveraged by JMX.

JMX defines only interfaces that are necessary for management. Thus, it is not a general-purpose distributed object system. [JAVA-JMX] The advantage of JMX is based on its capability to overcome the static nature of traditional network management systems. It recognizes the need for managing resources that appear, move, and disappear frequently. Additionally, the specification is not going to re-implement existing network standards but tries to incorporate them. The biggest drawback of this solution is its dependence on the Java programming language and on related Java technologies.

References

References in this document are constructed by the main authors name and the year of publication. For documents that are produced by an organization, the reference contains the organizations acronym and an abbreviation of the document title. Documents of project milestones are a combination of an acronym of the project name and an acronym describing the deliverable number.

References from the World Wide Web (WWW) are accompanied with the Uniform Resource Locator (URL) that points to the actual document in the WWW. Additionally, each WWW reference is provided with information when the author of this document has last visited the related document. It is most likely, that the content of the WWW document has changed since this last visit or that the WWW document is no longer available at all. All referenced documents from the WWW can be requested from the author.

- [Badach94] Badach, A., Hoffmann, E., Knauer, O.: *High Speed Internetworking: Grundlagen und Konzepte des FDDI- und ATM-Einsatzes*. Addison Wesley, Reading, Massachusetts, 1994
- [CORBA-MAN] CORBAMAN: *CORBA Management Agent Generic and NE specific information model*. Revision 1.1, March 6th, 2001
available at <<http://www.corbaman.com>> (last visited 03/14/02)
- [CORBA-TMN] OMG: *Interworking between CORBA and TMN Systems Specification*. Version 1.0, OMG Document 00-08-01, OMG, August 2000
- [Geihs01] Kurt Geihs: *Middleware Challenges Ahead*. IEEE Computer, Volume 34, No 6, June 2001
- [Funabashi00] Motoshiba Funabashi et.al.: *Development of Open Service Collaborative Platform for Coming ECs*. by International Joint Efforts. SSGRR 2000, August 2000
- [IETF-RFC1777] W. Yeong, T. Howes, S. Kille: *Lightweight Directory Access Protocol*. IETF RFC 1777, March 1995
- [Hegering99] Hegering et.al.: *Integriertes Management vernetzter Systeme*. 1. Auflage, dpunkt – Verlag für digitale Technologie, Heidelberg, Germany, 1999
- [ITU-X500] ITU-T Recommendation X.500 (1993): *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*. International Telecommunication Unit, Geneva, 1993
- [ITU-X903] ITU-T Recommendation X.903 (1995): *Information technology – Open distributed processing – Reference Model: Architecture*.
- [JAVA-JDMK] Sun Microsystems: *Java Dynamic Management Kit White Paper*. Sun Microsystems, Palo Alto, 2000
- [JAVA-JMX] Sun Microsystems: *Java Management Extensions – Instrumentation and Agent Specification 1.0*. Final Release, Sun Microsystems, Palo Alto, July, 2000
- [Magedanz99a] Thomas Magedanz: *Intelligent Network Evolution – Middleware Technologies for Open Service Architectures*. Habilitationsschrift, Technische Universität Berlin, Fachbereich Informatik, Fachgebiet für Offene Kommunikationssysteme (OKS), October, 1999
- [Magedanz01] Thomas Magedanz: *Enhancing Parlay with Mobile Code Technologies*. Proc. of the 6th IEEE Intelligent Network Workshop, IN2001, Boston, MA, USA, May 6-9, 2001
- [NMF-GB909] Network Management Forum: *SMART TMN Technology Integration Map*. Network Management Forum, Document GB909, Issue 1, April, 1998

- [Raymond95] Raymond, K: *Reference Model of Open Distributed Processing (RM-ODP): Introduction*. Proc. of the International Conference on Open Distributed Processing, ICODP'95, Brisbane, Australia, 20 - 24 February, 1995