RC 21769 (98003) (06/01/2000)
Computer Science/Mathematics

IBM Research Report

Managing Management Systems: A Distributed Applications Management Scenario

Alexander Keller

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Managing Management Systems:
A Distributed Applications Management Scenario

Alexander Keller
IBM Research Division, T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY, USA
Telephone: (914) 784 7593, Telefax: (914) 784 6183
E-Mail: alexk@us.ibm.com

Abstract

In today's networks where an arbitrary number of service providers needs to dynamically exchange customer- and system-related data, the integrated management of networks, systems and applications is a challenge. Management systems play a strategic role for enabling seamless interworking because they contain the management information that has to be shared between service providers in order to monitor and enforce service level agreements. However, this interworking is difficult to achieve: On the one hand, different service providers have chosen different management systems that are bound to specific, standardized management architectures and, due to the heterogeneity of the underlying frameworks, do not interoperate easily. Solutions for achieving interoperability between heterogeneous frameworks are one key factor towards integrated enterprise management and have been studied in the past. On the other hand, the provisioning of management instrumentation for the management systems themselves is a yet unresolved problem although the need for such a solution becomes apparent. Furhermore, management systems are a characteristic example of distributed applications: Developing a methodology for managing management systems helps also to understand which information and services are needed for the administration and control of distributed applications – an important and particularly complex field of investigation which is only starting to get attention.

The paper presents a novel approach to this problem by defining a management object model and appropriate instrumentation for management systems. It can be regarded as a step towards integrated enterprise management and is based on the Common Information Model (CIM) and the viewpoint languages and concepts of RM-ODP; their generic management models for distributed applications are then refined to handle the specifics of management systems. A CORBA/Java-based prototype implementation for managing management systems illustrates the applicability of our concepts.

Keywords: Application Management, Enterprise Management, Management Models, ODP, CIM, CORBA

1 Introduction and Motivation

The question "what are the characteristics of distributed applications from a management point of view?" is a current research topic [40, 7, 4]. Compared to other management areas, there is still a lack of common understanding how the static and dynamic aspects of distributed applications can be captured: Within other areas, there are models like the OSI Reference Model or the IP architectural concepts. Based on these and on many years of experience, there is a common understanding about what makes up a router, switch or other networking devices from a management point of view.

In order to describe these characteristics, we first need a framework for describing management objects representing (distributed) applications and their relationships. This framework must allow us to describe relevant aspects of any distributed application in a way that is suitable for all functional areas of management (Fault, Configuration, Accounting, Performance, Security – FCAPS). For example, it has to allow the specification of relevant aspects that arise e.g., from software management and distribution on the one hand

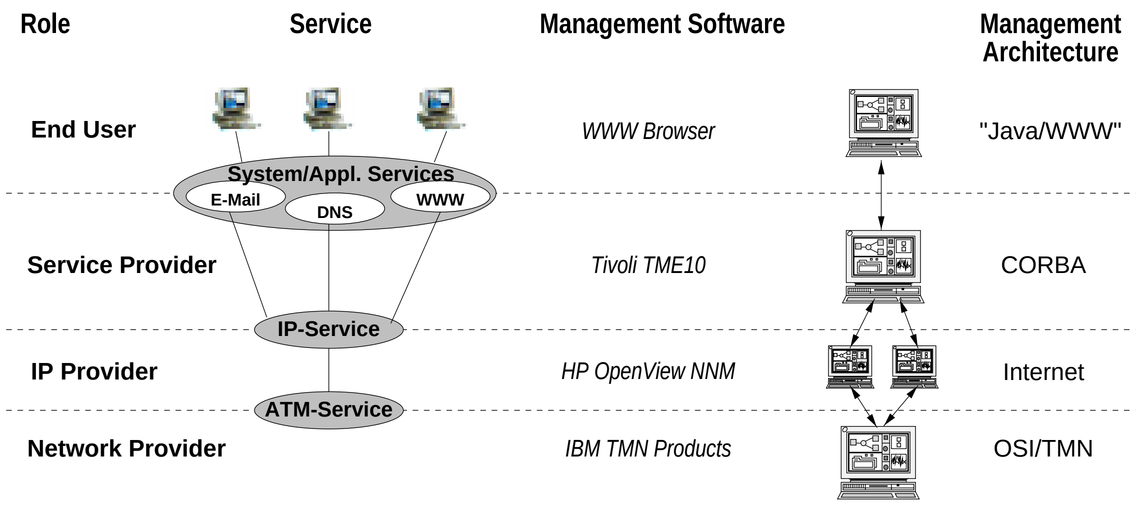


Figure 1: Service provider hierarchies require cooperation of management systems

and from the need to monitor the status of processes, their ability to communicate etc. and initiate corrective actions (in case of faults) on the other hand.

Integrated management solutions based on standardized management architectures aim to support the network and service providers' efforts in maintaining a high degree of quality and availability of their services while decreasing the cost of running the information technology infrastructure. Although providers can choose their management solution from a large number of implementations, the increasing complexity and heterogeneity of distributed systems still represents a major challenge for the Operation, Administration, Maintenance and Provisioning (OAM&P) of large-scale public and corporate networks.

From a management point of view, the situation has become more complicated due to the implications of telecom deregulation: Nowadays, an enterprise is not only free to choose between numerous IT service providers and network carriers but also needs to verify the QoS of the subscribed services. In addition, the role of the Internet Protocol (IP) as “lingua franca” for worldwide data communication and the layering of its standardized services and protocols (e.g., EMail (SMTP), Domain Name Service (DNS), World Wide Web (HTTP)) present opportunities of outsourcing IT activities for cost savings. This leads to layered service provider hierarchies as depicted in the left part of Figure 1 where a service provider is at the same time the customer of another provider: The provisioning of end-user services such as electronic mail, WWW and DNS therefore depends on the availability of the IP service which, in turn, requires e.g., an ATM service. The problem domain dealing with the fulfillment of *Service-level Agreements (SLA)* [43] between service providers and their users is currently a subject of ongoing research (see e.g., [6]).

In this paper, we will concentrate on the right part of Figure 1, i.e., the question how **Management Systems** of service providers and customers can cooperate effectively. Today, it is very likely that these systems cannot interoperate seamlessly because different service providers usually deploy management systems that differ not only by their vendor (IBM/Tivoli, HP, CA), but also by the underlying management framework (OSI/TMN, Internet, CORBA, Java/WWW, proprietary). Furthermore, when these systems were initially purchased, there was often no need to exchange management information with peer management systems operated by another authority. This situation has changed considerably: End users now want to check the quality of their subscribed services by retrieving data with WWW browsers from SNMP-based management systems, and SNMP-based management systems that initially controlled the local area networks of an enterprise need to exchange management information with TMN-compliant management systems surveying long-haul telephony links. As management systems are the point of control for services and networks, management policies have to be enforced and surveyed by them. All this makes management systems crucial for successful enterprise management. It is therefore not only necessary to (re-)configure management systems at runtime but also to control whether they are working properly. This paper presents an CIM/RM-ODP/CORBA-based approach to tackle this problem.

The above discussion shows that three questions must be answered to ensure the successful deployment of management systems in an open service market:

1. How can we achieve **Interoperability** between management systems, i.e., which mechanisms are needed so that they can exchange information with each other, even if they are based upon different management architectures? The integration of management architectures, i.e., establishing a so-called “Umbrella Management” is currently a large field of investigation [13]; section 3.1 gives an overview on promising approaches.

2. What management information is needed for managing management systems? What are the management requirements and how does an appropriate model of management systems look like? Section 3.2 focuses on the aspects of management information **Interchange**.

3. Which specific services are needed for managing management systems? How does an architecture for the **Interworking** between management services in a distributed object-oriented environment look like? Section 3.3 examines these issues.

Some questions relating to the design of a Management Information Base (MIB) for management systems are: How is the management model structured? Can we take object classes specified by standardized models as base Managed Object Classes (MOCs) for our model? In section 2, we elaborate on the specific characteristics of management systems and derive their requirements. We will then present in section 3 current methods and promising approaches for dealing with the overall problem of managing distributed applications in heterogeneous environments. This allows us to identify the main issues that must be taken into account when building management models for distributed applications and services. During our work, the notion of “application lifecycle” has turned out to be particularly useful for capturing the different phases of distributed applications. We describe these aspects in section 4 by means of an example, namely a management system that has been the target of our work. The identification of four aspects gives us the opportunity to evaluate several standardization initiatives with respect to their coverage of the different application phases. Section 5 focuses on the principles that guided us in designing our management model and how we made use of RM-ODP and CIM concepts. We also describe the application of the Unified Modeling Language (UML) and the impact of CASE technology on the design of the object model. In addition, we provide an overview of our implementation and how general-purpose distributed object technologies such as CORBA and Java were used for the prototype. Section 6 concludes the paper and presents issues for further research.

2 Requirements Analysis

This section analyses the properties of management systems and the requirements regarding the necessary management information and management services, respectively. It provides the conceptual background and helps to identify important features needed for the definition of the object model for management systems, described in section 5.

2.1 Characteristics of Management Systems

The way how management systems are deployed today is by establishing well-defined, central points of control, termed *management platforms* (see [11] and [5]). As outlined in previous studies [44], platforms tend to become bottlenecks in large networks and a solution to this problem is to distribute their functionality (e.g., topology functions, event filtering mechanisms, logging facilities) in order to reduce the load on the central management system. The standard solution to address this issue is by introducing subordinate management systems acting in the role of **Mid-Level Managers (MLMs)**; they are located close to the managed resources and execute management functions on behalf of other management systems. Their main purpose is to condense raw data stemming from the resources into meaningful management information that can then be used by another management system. This may also include the caching of frequently accessed information such as topology data of the underlying systems. These properties obviously reduce the amount of overall traffic that is sent through the network for management purposes. MLMs are an effective mechanism for structuring networks into domains and are used for establishing management hierarchies. An extension of this concept is to delegate (and withdraw) management functionality to MLMs at runtime [12]. This delegation may either be initiated by the MLM itself (pull model) or by another management system (push model). The interworking between management systems of different service providers as described in section 1 can be seen as a special case of interactions between different MLMs. Consequently, the above mentioned principles apply also to independent management systems operated by different authorities.

Further complexity is introduced by the heterogeneity of the deployed management architectures.**Management Gateways** are a convenient mechanism for achieving interoperability between heterogeneous management systems. They are located on the boundaries of different (architectural and organizational) management domains and are perceived as Mid-Level Managers by other management systems. As management gateways are aware of the architectural specifics of the underlying resources, it therefore makes sense to provide them with the same management functionality as MLMs. Such an enhanced management gateway could e.g., condense several SNMP-traps into one CORBA-event instead of translating every trap into one event and leaving the filtering of these events to the management system one level above in the service provider hierarchy. This implies that providers need an instrumentation for these systems in order to present a unified view on their management data to customers; the amount of accessible information is usually specified in a service contract.

Although some existing management frameworks provide mechanisms of inter-manager-communication (OSI/TMN: x reference point, SNMP: inform-pdu), it is currently not clear what an appropriate management model for controlling management systems should look like and what actions may be initiated by a management system on behalf of another one. We will now first describe what kind of management information ought to be present in an object model for managing management systems and will then move on to the required management services, i.e., the functionality that management systems may provide for interacting with peers. The identification of management information and management functionality was based on a use case analysis applied to typical management scenarios. For the sake of brevity, we can only describe a small part of the amount of management instrumentation identified in our analysis. The complete set is described in [27].

2.2 Management Information

As with any other kind of managed resource, generic configuration management information (manufacturer, product name and version, installation date, etc.) and properties relevant for fault management (support contact, time since last restart and its usage, operational and administrative states) are relevant for managing management systems. If a management system supports delegation, information about the supported scripting languages or the version of the execution engines should be available. From the inventory perspective, it is also necessary to provide an overview of the installed components, their versions and their states (e.g., the installed system modules, the database and the management applications, the type of available delegated management functionality) and the corresponding process names.

The gathering of licensing data relevant for accounting management such as the kind of product license (nodelocked, domain-bound, floating), the maximum and the actual amount of management system users is necessary because many commercial management system products are equipped with license servers. A high amount of rejected requests due to an insufficient number of licenses indicates the need to acquire additional licenses. This may either concern the number of concurrent users of the management system or the number of managed nodes. Appropriate counters need to be provided.

Security is a major concern in distributed environments where several management systems are acting as peers: there is a need to provide information about the management domains a management system is involved in and the agents, MLMs and management gateways it is responsible for. If a management system is able to configure another one, every involved system needs to maintain on the one hand view-related access control mechanisms for protecting itself against unauthorized access and, on the other hand, information about its own capability set, i.e., what kind of interactions with peer management systems are permitted.

Performance-related information includes parameters for configuring MLM caching properties such as the cache size or the maximum aging time for the cached data. Counters for the time a management system takes for serving a request and the number of requests per (user-configurable) interval may indicate performance bottlenecks requiring additional MLMs.

Additional information needed to control management gateways encompasses the architectures the management gateway translates, the names and versions of the management information models and protocols supported. Of equal importance are counters for (successful, erroneous) translated protocol data units and information related to the mapping of managed object references.

2.3 Management Services

This section will sketch a subset of the overall management functionality that is needed for management systems in order to interwork properly.

Configuration management services provide the ease of introducing new management systems into distributed environments by dynamically assigning them configuration profiles covering domain affiliation and polling intervals. A management system therefore needs to provide functionality for downloading initial configuration profiles to other management systems. If delegation is supported, the usual operations that apply to delegated management functionality (start, stop, suspend, resume etc.) should be present.

Of major importance for fault management are basic operations such as verifying whether all the components of a management system are running and services for (re-)starting or stopping either the complete management system or selected components should be available. Checking the consistency of the management system database and the execution of maintenance (e.g., backup and restore) and error detection tasks need also to be done at regular intervals that may require scheduling mechanisms. In case of errors, the event and error logs maintained by a management system provide a rich source of information for determining the probable cause of failures; consequently, facilities for browsing and searching these logs according to different criteria should be available.

The configuration of event forwarding mechanisms, the initiation of management operations on peer management systems and the registration of a management system for specific events require security services to ensure that no security policy is violated.

Services related to performance and accounting management include the generation of reports on resources administered by a specific management system (statistics, system parameters filtered according to different criteria such as throughput, delay, load and usage) and should therefore be made available to network administrators. In addition, parameters such as the ratio of application messages to network packets, the percentage an I/O device is busy, the queueing delay while a process is ready and waiting for CPU are useful to derive performance metrics for a (management) system. Note that the usability of these parameters depends to a high degree from the policies defined by the organization that deploys the management system: The selected systems, the probing intervals for the aforementioned metrics and the way these metrics are combined reflect the policies of an enterprise. It is therefore necessary to provide *individually configurable management functionality* to create meaningful information from available management data. Section 3.3 elaborates on this aspect.

3 State of the Art in Distributed Application Management

Early research in application management has focused primarily on solutions based on the OSI management architecture [15]; the emerging Common Object Request Broker Architecture with its CORBAservices allowed the exploitation of mainstream object-oriented technologies for systems and application management [37, 36]. However, the considerable delays in the adoption of appropriate CORBAservices postponed the availability of CORBA-based management [26]. Meanwhile, vendors have developed their own, proprietary infrastructures to satisfy the rapidly increasing demand for the end-to-end management of enterprise

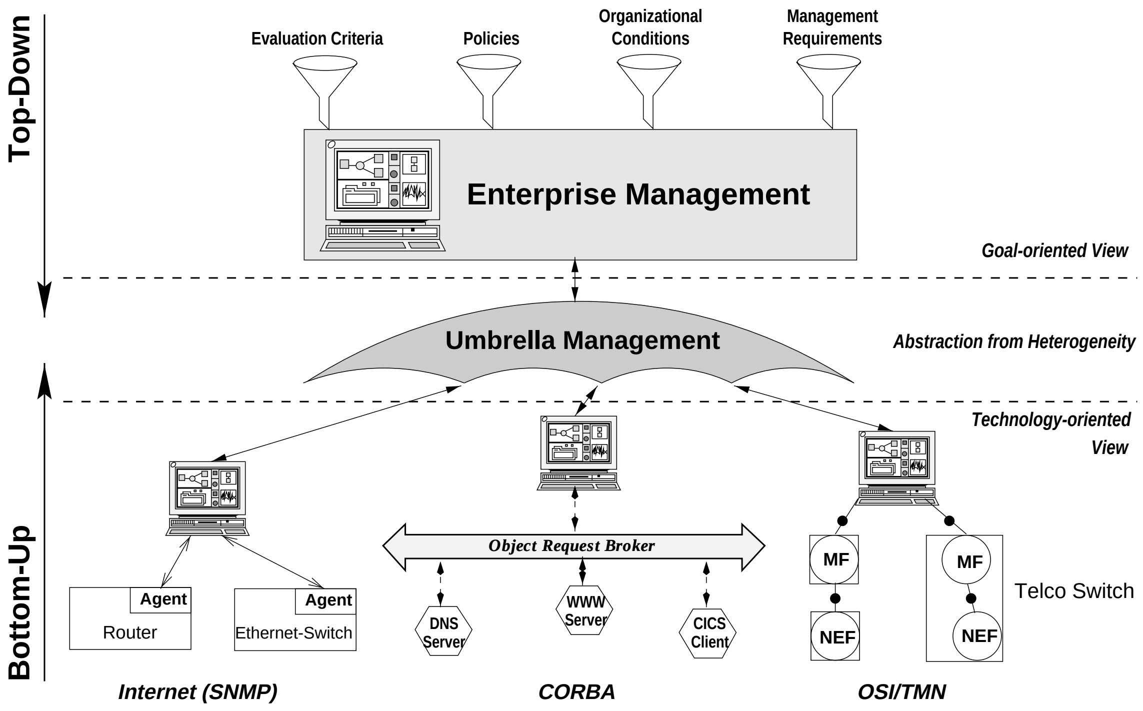


Figure 2: Umbrella Management as a basis for Enterprise Management

applications and services [17]. All these approaches have in common that they assume a homogeneous management infrastructure; however, this is usually not the case, as the previous sections have pointed out. This section describes the prerequisite steps and the essential building blocks for distributed application management.

3.1 Achieving Interoperability: Umbrella Management

A major objective of **Enterprise Management** consists in presenting a unified, all-encompassing view on networks, systems, services and applications. It is oriented towards the providers' goals and gives them the ability to control any kind of managed resource according to their management requirements, policies and organizational conditions. Enterprise Management follows a top-down approach and should therefore not be constrained by technical intricacies. On the other hand, the amount of management frameworks providers have to deal with is already determined by the nature of the managed resources: LAN components usually have SNMP agents while telecommunication components are managed according to the OSI/TMN framework; distributed applications and services might provide a CORBA-based management instrumentation. **Umbrella Management** has to cope with the heterogeneity of Management and focuses on the technology-related aspects of cooperation between entities involved in the management process. Its objective is to abstract from the heterogeneity of the underlying management architectures by defining means of interoperability. It is therefore fundamental for integrated Enterprise Management (see Figure 2).

There are basically three Umbrella Management strategies for achieving interoperability between systems located in different architectural domains (see also [24]):

The first approach consists in the integration at the resource level, i.e., the managed systems support more than one management protocol; they are equipped with **Multiarchitectural Agents**; [31] is the first paper describing such an implementation. However, this approach is usually difficult to pursue for the following reasons: Often, agents are used to perform monitoring of simple network devices such as hubs or bridges. They are usually built into the firmware of the device and should thus have a small footprint;

the implications are that these agents can neither be enhanced to support another management protocol nor should they introduce additional overhead.

An alternative is to place the burden of integrating the different architectures on the management system. Such a **Multiarchitectural Manager** supports a set of management protocols, which are implemented onto the platforms' communication stack. Thus, conversions between different management protocols are not necessary. The transformation of the management information descriptions is often handled by tools bundled with a management system, like MIB compilers, and therefore need not be handled by the developer. In addition, the service APIs of management systems can easily be accessed, thus yielding the opportunity of reusing the large amount of platform services, such as event filtering, topology, threshold monitoring or resource discovery (see also section 5.3). On the other hand, these APIs are specific to a concrete management system product: the portability of interoperability solutions based on this integration paradigm is often restricted. An example of a multiarchitectural manager is IBM *NetView* (SNMP-based manager) with the *NetView TMN Support Facility* (for OSI/TMN).

The third solution is the **Management Gateway** approach. It is then possible to manage services, systems and networks in different management architectures from a single point of control, as demonstrated in [32] and [39]. Standardized mappings between the OSI, Internet and CORBA information and commnication models have been developed by the *Joint Inter-Domain Working Group (JIDM)* [21] and academic institutions [34]; commercial implementations are emerging [14]. Our experiences with building management gateways [28] have shown that they represent powerful instruments for bridging the gaps between different management architectures because neither the managing nor the managed systems need to be modified. This is a crucial feature for the interworking between service providers as described in section 1. However, it has to be recognized that gateways are very complex systems and – due to their high relevance for enterprise management – need to be managed as well; in section 5, we describe why we consider management gateways as a special case of management systems and show what their management instrumentation may look like.

3.2 Ensuring the Interchange of Management Information Models

The very first attempts addressing the issue of distributed application management have been carried out by the Internet Engineering Task Force. However, the Application MIBs (*applMib* [23] and *sysApplMib* [30]) of the Internet management architecture are mainly focused towards a specific operating system and the issues with the object-based information model compromise their usability: While the System Application MIB reflects basically static application information and the status of the associated processes, a large amount of information contained in the Application MIB deals with the issue of indexing the tables containing the effective management information according to different criteria.

The main motivation for our approach is as follows: As stated in section 1, management is heterogeneous: several new frameworks have emerged during the last few years, some of them emphasizing on the information aspect (such as CIM), others on the communication infrastructure (such as CORBA and the Java Management Extensions (JMX)). Our goal is therefore not to invent an additional architecture but, instead, to combine standardized frameworks: we make use of existing management information models (CIM and RM-ODP), use a widely deployed notation (UML), and implement our work on top of a Java Object Request Broker. This leads to a combination of proven technologies and allows us to build a unified framework covering the analysis, design and implementation stages of our work.

Another fundamental assumption is that we consider management itself as a distributed application; the consequence that the distributed application "management" itself needs to be managed has already been motivated in section 1. Another implication is that we can apply already defined and standardized models for (general-purpose) distributed applications to the management of management systems. We will show in section 5 how we used the RM-ODP to obtain generic management object classes for application management; we will use these as base classes for building the inheritance hierarchy of our object model for management systems.

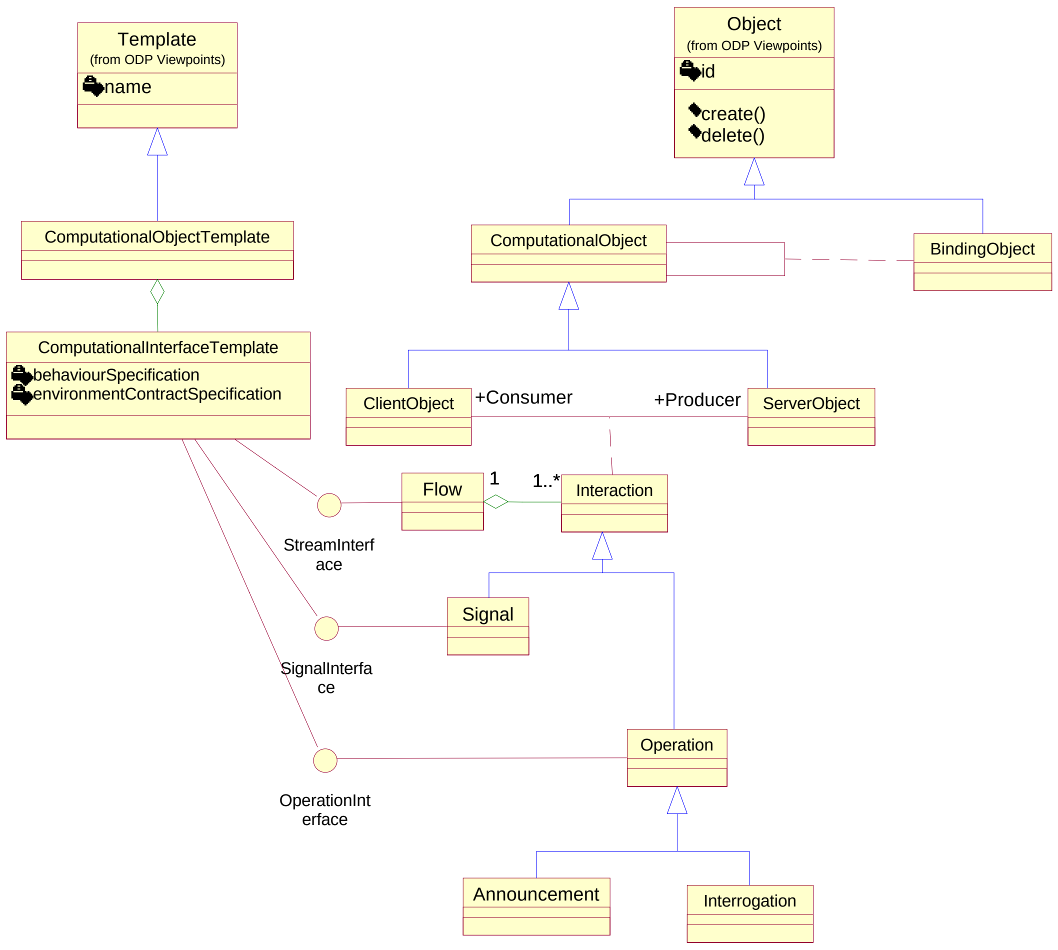


Figure 3: Object classes specified in the RM-ODP Computational Viewpoint

3.2.1 Reference Model of Open Distributed Processing (RM-ODP)

In earlier work [29], we have identified the RM-ODP [19] as particularly suitable because it provides the necessary terms and notions to cover the different aspects of distributed applications, particularly at the runtime stage. It has therefore been taken as the basis for deriving our generic application management instrumentation.

Analyzing the ODP concepts from a management point of view reveals that the *Computational* and the *Engineering Language* are of highest relevance for our purposes because their concepts define a considerable part of the resources that have to be managed. In contrast, concepts from the *Information Language* cover the semantics of information processing and are only of secondary concern for (technical) management. Since integrated management should abstract from the technical realization of the managed resources as far as possible, this also holds for the *Technology Language*. Furthermore, the lack of applicability also applies to concepts from the *Enterprise Language*; however, it has to be reconsidered if advanced management concepts such as policy-based management are deployed.

Concepts of the **computational** language allow the functional decomposition of an ODP system into objects interacting at interfaces. Therefore, MOCs based on them, e.g., *operation*, *signal* or *binding* are important for describing the relationships between managed objects. No other management architecture

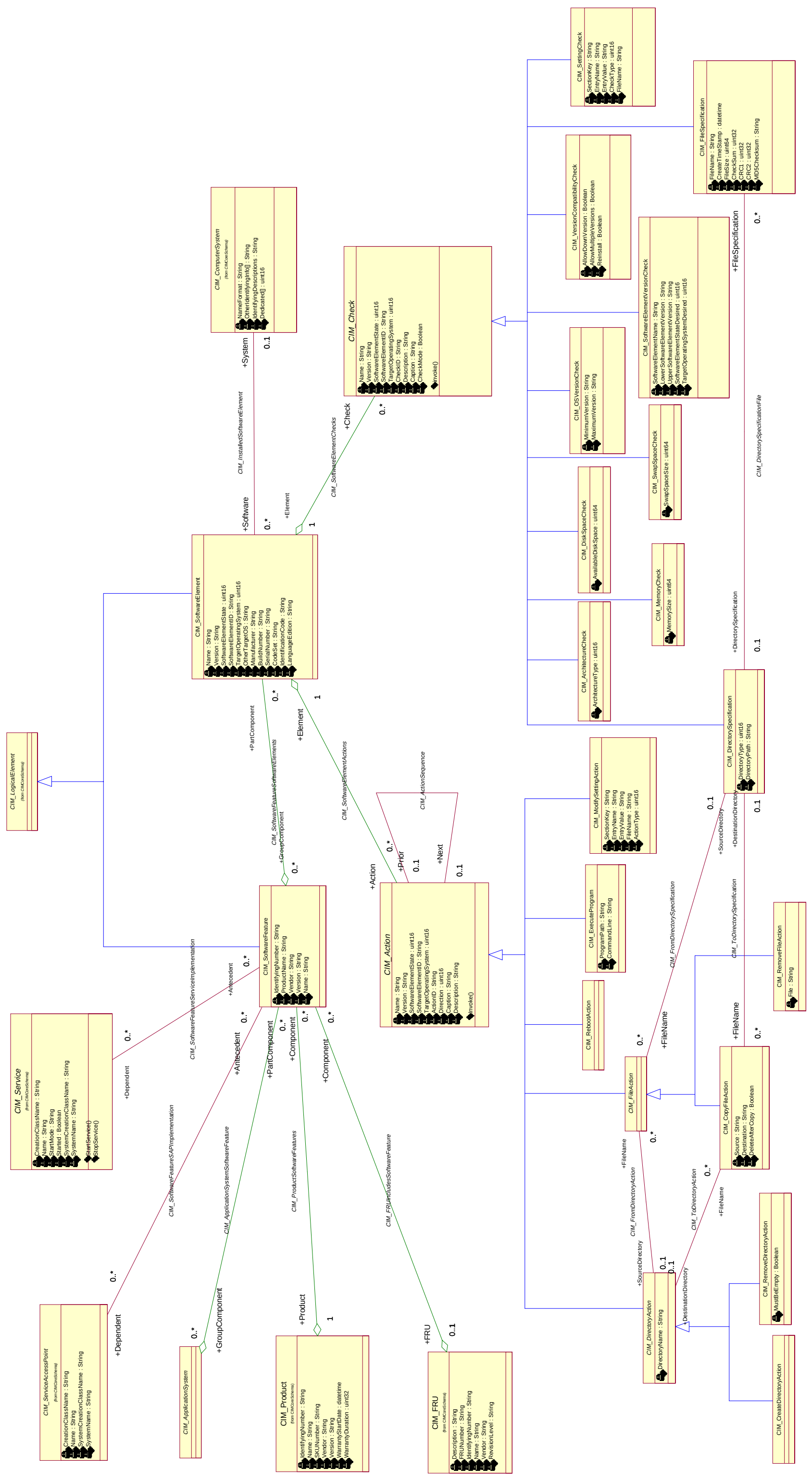


Figure 4: CIM Application Schema (without association classes)

provides similar concepts although they are definitely needed in order to capture the dynamic relationships between distributed applications. The object classes defined in the computational viewpoint are depicted in figure 3.

An **engineering** specification defines the mechanisms and functions required to support distributed interactions between objects in an ODP system. Concepts of the engineering language e.g., *node, capsule, cluster, channel* etc. or, respectively, the MOCs derived from them, support the monitoring of processes, the connections between them etc., i.e., dynamic or runtime aspects of application management. An example for this is the concept of capsule as defined in [20]: "A configuration of engineering objects forming a single unit for the purpose of encapsulation of processing and storage"; this definition maps to the notion of a process running under the control of an operating system.

3.2.2 Common Information Model (CIM)

The DMTF *Common Information Model (CIM)* [9] follows an object-oriented approach and defines managed resources as object classes with properties and methods that can be further refined by means of strict inheritance. In order to circumvent multiple inheritance, CIM makes extensive use of various types of aggregations and relationships: These relationships are modeled as association classes. The *Core Schema* defines basic terms as abstract classes such as service access point, service, product, system, logical device etc., and provides a means for associating context (setting, configuration) with them. It is mandatory that any CIM resource implements at least the core and one of common schemas [8]: The *System Schema* is one of the various common schemas and refines the root classes of the core schema in order to deal with jobs, hosts, operating systems, processes, threads and file systems. The *Application Schema*, as another example for a common schema, refines the core schema w.r.t. distributed applications: It defines units of deployment (software element), units of component management (software feature) and allows the grouping of features in a business system (application system). It is depicted in figure 4. In addition, various checks (OSVersion, VersionCompatibility etc.) and actions (Reboot, ModifySetting) are defined. Finally, the *Distributed Application Performance Schema (DAP)* relates the definition, the metrics and the logical element that is instantiated to a so-called "unit of work". In total, the amount of managed object classes being defined in CIM comes close to 250. It is therefore fair to say that even if CIM is still evolving, it represents a solid basis for integrated management. The exchange of management information between managed resources, management systems and management applications is done by encoding the CIM object descriptions in XML and transferring them over HTTP. However, the immaturity of current XML, DTD and XSL development tools does not allow us to make use of the recently specified CIM/XML mapping: Currently, we use CIM entirely for modeling purposes.

CIM uses the *Unified Modeling Language (UML)* [42] as notation for specifying the various schemas, thus leveraging existing general-purpose development and software engineering tools. It is possible to transform the MOCs – derived from the core and common schemas – into widely used programming and description languages (OMG IDL, C++, Java, XML). For our modeling, we have used a UML-compliant CASE tool [35] that meets these requirements.

3.3 Interworking between Management Services

3.3.1 Traditional (two-tier) Management

This model defines two entities involved in the management process: the manager processes all the management information according to the needs of a customer, and the (dumb) agent exposes **stateless** management information, i.e., raw data (counters, gauges, names etc.) captured from a resource. Note that the term stateless does not imply that the resource state is not contained within the management information (the state of a given resource and its part is obviously not only readable but also modifiable through the instrumentation, e.g., by means of a reboot operation): Instead, stateless management information means that there are neither implicit nor explicit temporal relationships between different items of management data; i.e., *there is no notion of history*. In general, an agent is unable to compare the values of a specific attribute over different time periods in order to determine whether a given attribute changes its value[1]. "Classical" SNMP resource agents therefore do not calculate averages or medians because this would require them

1. to have a notion of history (meaning that they must store management data of previous time periods), and,
2. to accept, check and store a given time interval defined by the manager in order to compute data such as "invalid packets per second during the last hour". Note that a counter "invalid packets per second" alone does not make much sense because it is unclear *which* seconds should be actually considered for the analysis.

Both items would violate the principle of stateless management information.

Obviously, administrators are primarily interested in information related to time: It is not particularly useful to know the current value of the counter "received IP packets"; the benefit of this counter becomes only apparent if it is correlated with time and/or other resource attributes. The fact that, e.g., the average load of the incoming packets during the last 3 hours was always above a given threshold is an important indication that the resource capacity is either insufficient or the network topology unbalanced.

This kind of information, derived from the data provided by the agent, highly depends from a notion of time and history and is termed **stateful management information**; the IETF Remote Network Monitoring (RMON) MIB is the first example of a MIB that computes stateful management information and is not bound to specific resources. Stateful management information can be regarded as *customized management information according to the goals of the management system operator*. As the observation intervals are not known in advance (some administrators may choose intervals of 5 minutes, others are only interested in daily statistics), this information is generally not computed by the agent but needs to be processed on the manager side. It is usually done by traffic/usage analyzers being part of management platforms. This approach has been frequently criticized; the main argument (among others) against it is that doing all the processing on the side of the managing system generates an enormous load if several thousands of agents are involved. RMON probes and mid-level managers help to reduce this load but cannot eliminate this problem.

3.3.2 Management Middleware: Introducing three-tier Management

A pragmatic way to obtain stateful management information (tailored to the specific requirements of a given customer) while maintaining the scalability of the overall management system is to separate the necessary functionality from the manager and distribute it across the network: Distributed object architectures such as CORBA and (Enterprise) JavaBeans allow the implementation of *configurable* management services in a way that they can be replicated and migrated according to a specific network and application topology. This is particularly important if the management of distributed applications is being considered because their number is usually at least two orders of magnitude higher than the number of network and system resources. It is fair to say that 3-tier Management splits the functionality of a "traditional" Manager in two parts: user interaction and management middleware.

Appropriately parameterized management services thus allow to generate stateful management information from stateless resource management data "on the fly", i.e., without having the need to store data persistently for a longer interval than really needed. Figure 5 depicts such a three-tier management architecture. The Gartner Group defines Middleware as the "glue between clients and servers" [10]; we define **Management Middleware** as *the services between managers and agents that help to make management more scaleable by bringing the (pre-)processing of management information closer to the resources.* It combines and enhances the management data exposed by the agents to meaningful management information and presents it to the manager. In addition to answering queries issued by the manager, the middleware can be configured by the manager in order to reflect the managers' policies.

[1]While this statement is appropriate for SNMP-based Management, the OSI/TMN-Management provides an "Attribute Value Change" notification. However, it is unclear whether the OSI/TMN-compliant management platforms deployed in large environments make use of this feature because it tends to increase the number of notifications.

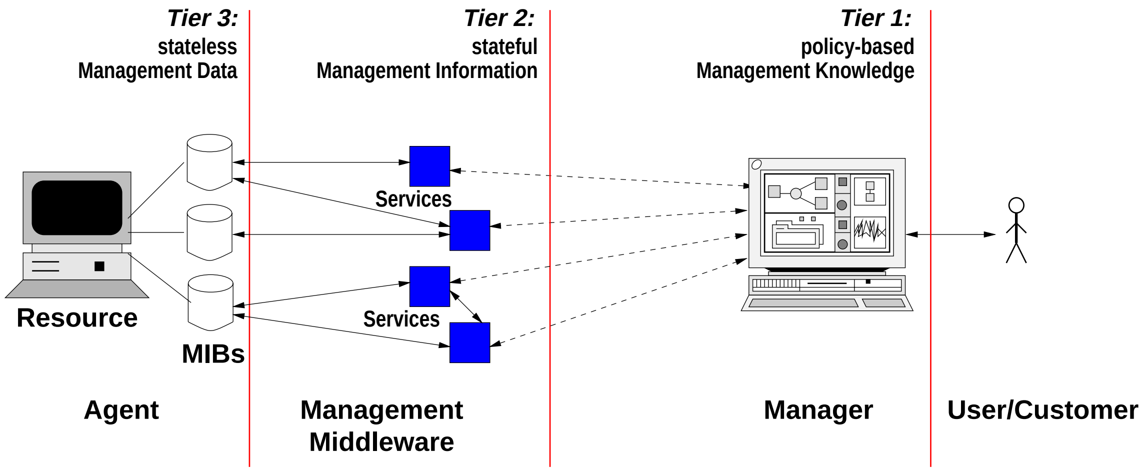


Figure 5: Management Information available at the Interfaces of a 3-tier Management Architecture

However, it should be noted that today's management platforms (such as NetView and Enterprise Console) put both tier 1 and tier 2 in a single system because the management middleware (topology manager, status monitor, object database) is bound to (and unseparable from) the core product. Implementing these systems on a distributed object-based infrastructure yields the benefit of distributing the management middleware "over the network", thus separating tier 1 and tier 2.

There are basically three different types of services that are part of the Management Middleware:

1. generic management functionality common to a wide range of managed resources. Typical examples for these are the services defined by the *OMG Telecom Domain Task Force* (Notification, Logging, Topology etc.) and the OSI/TMN *Systems Management Functions* (policy, log control, event management, management knowledge, event filtering and forwarding of selected information, storage of resource data for caching purposes etc.). A recent approach to defining generic management services in a CORBA environment is the work of the TeleManagement Forum *Application Component Team (ACT)* [41]; an exhaustive list of management requirements is provided in [2].

2. Appropriately modularized problem management knowledge captured from experts; the management services can therefore be regarded as "canned knowledge". Examples are: checking the state of services on which an application depends; verifying whether a filesystem is mounted if a webserver continuously encounters errors while trying to access related documents etc.

3. services that are now implicitly part of management systems (see above): These can be tailored towards the needs of a specific provider so that the resulting stateful data can be compared to the individual policies of the provider. Typical examples are: The ratio of application messages to network packets during the last hour, the average percentage of busy I/O devices on a given server farm, the average queue length of the print spooling system at 11 am; [47] provides some more parameters that require parametrization because different customers have different needs.

4 Aspects relating to the Lifecycle of Management Systems

In order to structure the requirements of management systems and gateways it is helpful to characterize them according to the different aspects how they are perceived. Figure 6 depicts the four different aspects that we have identified during our research:

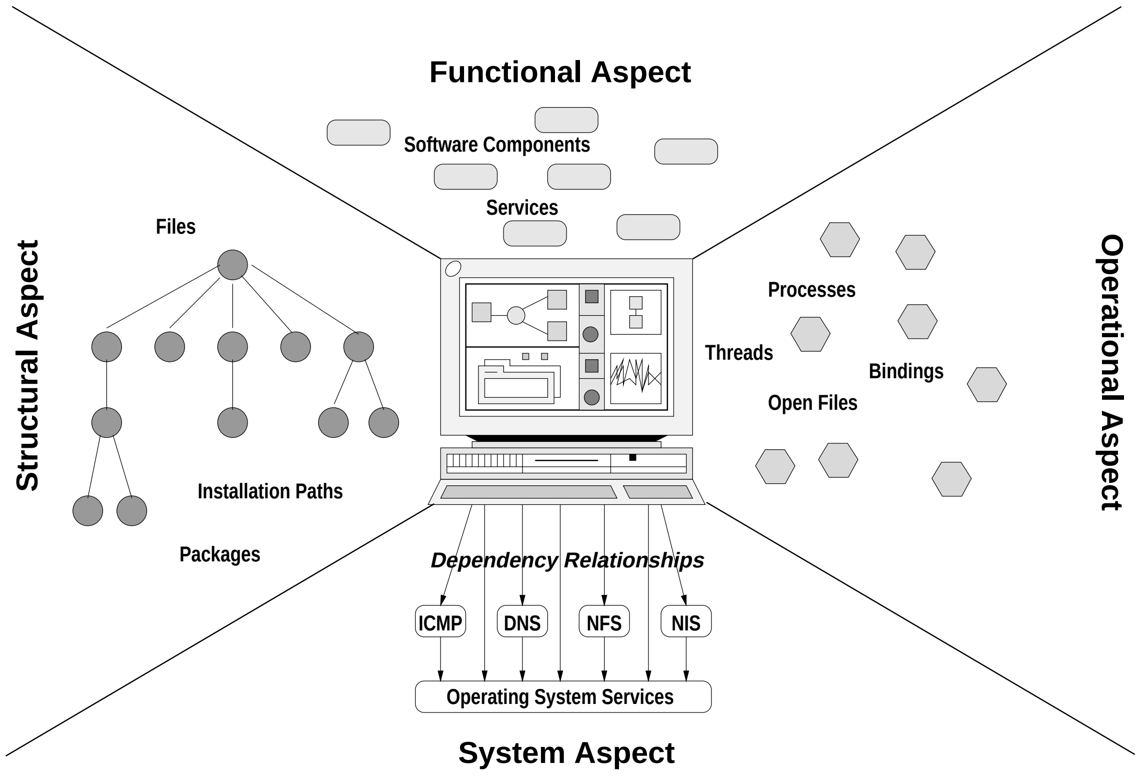


Figure 6: Aspects of Management systems

- The **functional aspect** describes a management system as an installable software package (e.g., *NetView for AIX*), which consists of several components providing distinct services. The AIX *System Management Interface Tool (SMIT)* gives the following list of the NetView components: *Base System*, *Database*, *Developer Supplement*, *Online Documentation* etc. These components can be further refined to reflect the actual services they provide, such as configuration manager, performance monitor, topology module, event handler, status monitor, user interface, communication component, MIB browser. The functional aspect addresses the question *"What does the application do and which services does it offer?"* It should be noted that these services are not only provided to end users but to other (eventually remote) services as well.

- If a management system is regarded as an executable program system, the focus lies on its **structural aspect**. It is perceived as a set of files, which contain either the services or configuration files, help functions or management data related to the administered resources. Relevant management information are Packages, Files, Installation Paths etc. Software distribution and installation tools make use of this information. The structural aspect deals with the question: *"Which components have to be installed?"*

- When the management system is instantiated (**operational aspect**), it is perceived by a set of communicating (UNIX-) processes (*ovtopmd, netmon, pmd, nvsecd*), by a list of open files and established client/server bindings. The purpose of the operational aspect is to answer the question: *"How well does the system run and perform?"*

- The **system aspect** recognizes the fact that a management system cannot be regarded as an isolated entity, but is dependent from operating system functions (such as disk and swap space) and networked services such as ICMP, DNS, NIS and NFS. The latter is important because the NetView database might reside on a remote host, thus a failure of the remote host implies a failure of the management system. The question *“Which underlying services are required and what are the dependency relationships?”* is the purpose of this aspect.

While the first three items basically reflect the main stages of the software lifecycle (design, installation, runtime), the system aspect does not fit into this categorization. However, it is important to address this issue because today, the runtime status of an application has to be determined by invoking operating system routines due to lack of proper instrumentaion. In addition, vital application dependency information is usually kept and maintained by the operating system in system configuration repositories (and not necessarily by the application itself); it can be retrieved from the repository by means of operating system calls. The methodology for achieving this and a detailed example are described in [25].

4.1 Relationships of management systems to the System and Network Infrastructure

Management systems usually run on general-purpose operating systems like UNIX or WindowsNT and are therefore sensitive to errors that occur in the underlying operation system and the communication infrastructure provided by networked services (e.g., the *Domain Name System (DNS)*, the *Network File System (NFS)* and the *Network Information System (NIS)*) and protocols. Consequently, commercial management system implementations dedicate a high degree of their management instrumentation to the monitoring of the underlying system. While this may be necessary, in our opinion it is not the primary concern of a MIB for management systems to control not only the management system itself but also the underlying environment. Thus, we restrict ourselves to the instrumentation of the management system itself and consider the instrumentation of operating systems (CPU usage count, user quota, disk and paging space etc.), networks (latency, timeout errors, network buffer size, packet and frame errors) and underlying networked services such as DNS, NFS and NIS as out of scope for this paper. Nevertheless, we have designed object models and implemented the corresponding agents for UNIX systems and networked services. Readers interested in these agents are referred to [33]. The development of these agents has been done according to the approach described in section 5, too. On the other hand, operating system repositories such as the IBM *Object Data Manager (ODM)* [16] provide a vast amount of static management information pertaining to management systems such as the names and versions of the software modules, their installation paths, their prerequisites and dependencies. As these tools are accessible through APIs we were able to retrieve this management information.

Note that although these aspects are defined in the context of management systems, they also apply to any other kind of distributed application. Management systems and gateways can be regarded are a characteristic example of highly distributed applications.

4.2 Related Work

Several models and standards exist which are relevant for application management; they provide a wealth of management information. These are:

- The IEEE 1387.2 *POSIX Software Administration Standard* [18] focuses on management information for software distribution and installation. The aspect addressed by this specification is the **structural** one.

- The *Distributed Software Administration (XDSA)* Specification from the Open Group [45] extends IEEE 1387.2 and therefore also deals with the **structural** aspect. More specifically, it is targeted towards software distribution/deployment/installation, defines several commands (swinstall, swlist, swmodify etc.) and a software definition file format with several attributes. However, XDSA does not deal with *instantiated* applications and services; therefore it does not have any means of representing them at runtime.

- The ISO *Reference Model of Open Distributed Processing* [19] defines terms and notions for describing distributed applications and their components. It can be used for the **operational** and **system** aspects (see also section 3.2.1).
- The Tivoli *Application Management Specification (AMS)* [1] is particularly helpful for the **system** and **structural** aspects.
- The *Application Response Measurement (ARM) API* [3] and its recent extensions [22] focuses on the performance management part of the **operational** aspect.
- The CIM [9] System and Application Schemas address primarily the following aspects: **structural** and **system**.

It can be seen that none of the above specifications is able to cover all the four aspects of distributed applications. We have chosen to combine two of them (RM-ODP and CIM, see below) to establish an application management model that is able to deal with all the aspects.

Today's applications almost never provide management instrumentation; the management information can be partly obtained from various (proprietary) APIs but, often enough, this information is only accessible for read-only purposes. Thus, in the best case, one can build a rudimentary monitoring for *a specific* application; active management does not happen yet.

These problems stem from the fact that it is yet unclear what the generic properties of applications are, i.e., a subset of management information common to all kinds of distributed applications. On the other hand, CIM and RM-ODP specify the components and properties of distributed applications in a generic way and therefore provide guidelines on

1. the structure of a broad range of distributed applications, and,
2. what can be reasonably assumed regarding the minimum set of management information any distributed application may have.

Our idea therefore consists in applying standardized architectural models for (general-purpose) distributed applications to the *management* of distributed applications. We will show in the following section how we used concepts from RM-ODP to obtain generic management object classes for application management; we will use these (together with the CIM schemas) as base classes for building the inheritance hierarchy of our object model.

5 An Object Model for Managing Management Systems

Seamless interworking between different management systems requires that the management information and the appropriate management services identified in the previous section are defined in a way that they can be accessed by peer management systems. Consequently, this section will focus on establishing an object model (i.e., a MIB) for management systems. In section 5.1, we show how we achieved a unified representation of management information common to all different kinds of entities involved in the management process. We then describe in section 5.2 how this management information can be refined w.r.t. the needs of specific management systems and how this process is eased by CASE tools. Section 5.3 gives some insights into our prototype implementation.

5.1 Defining the Base Classes of the Inheritance Hierarchy

Until recently, the inheritance hierarchies of object-oriented management frameworks were usually shallow and the degree of reuse was limited because the predominant part of relevant management information has often been specified in resource-specific classes. Consequently, the base classes contained only a restricted amount of information. On the other hand, one must recognize that – in general – different distributed applications share a lot of common properties: e.g., they are delivered as packages that consist of modules implementing a service, which is instantiated as a process. What we need is a framework that allows us to describe the relevant aspects of any kind of distributed application (in our case: management systems) in a way that it is suitable for management purposes.

The overview of CIM in section 3.2.2 has shown that this architecture differs fundamentally from existing approaches: CIM makes a serious attempt to leverage the power of common off-the-shelf object-oriented technology to define semantically rich object models suitable for today's complex distributed systems. However, the relationships between the different CIM schemas are not easily visible because many classes defined in a schema have relationships with classes defined in other schemas. Inheritance relationships spanning up to five class levels and the excessive use of association classes (the ComputerSystem class of the system schema alone is associated with eight other classes) introduce a considerable amount of complexity and make the graphical layout of the diagrams alone a challenge. The use of a new notation, the Managed Object Format (MOF), as an intermediary representation for management data exacerbates this problem. On the other hand, a developer is able to take advantage of CASE technology, thus making the task of deriving new, specific managed object classes from the existing CIM schemas relatively easy.

On the other hand, CIM currently misses object classes that capture the runtime behaviour of applications: entities such as bindings, queues, requests or signals are not yet present although the CIM Application Management Working Group has started to define appropriate runtime extensions for the application schema.

The ODP computational language helps to fill this gap because it defines various kinds of interactions between clients and servers (see figure 3 in section 3); its structuring rules specify useful constraints that help to clarify the semantics of the involved objects ("a computational object offering a stream interface generates flows that have producer causality in the interface's signature [...]" [20]).

The ODP engineering language, in contrast, does not add much value to CIM: there is a considerable overlap between the RM-ODP engineering viewpoint and the CIM system schema because both decompose applications in entities visible from a systems management perspective (e.g., threads, processes, operating systems, nodes). It is fair to say that the classes of the CIM system schema are a superset of the ODP engineering language concepts, thus making the latter obsolete for our specific purposes. In addition to ODP, CIM defines a large amount of class properties and methods that cover a large degree of system administration tasks.

Our solution therefore consists in following a "best-of-breed" approach by combining the ODP computational language with the CIM core, system, and application schemas. Together, they form the base classes of our object model for management systems and gateways. Thus, many required attributes and methods are already defined at the top of the inheritance hierarchy: Important characteristics relevant to software packages are present; in addition, instrumentation for installing and updating them is specified. Descriptions of services and their technical realization (as processes) are available. Services and applications can be started, stopped, suspended and resumed; dynamic bindings and various kinds of interaction can be represented. This covers a large amount of MIB-instrumentation that is usually defined only on lower, i.e., resource-specific levels (e.g., every SNMP group or table contains `Name` and `Description` attributes). Consequently, we can then guarantee a minimum amount of instrumentation commonly applicable to all kinds of distributed applications.

5.2 Representation of MOCs for Managing Management Systems

As mentioned before, the different CIM schemas and the model of the ODP computational viewpoint form the base classes for our management information and services (described in sections 2.2 and 2.3) become attributes and methods of these enhanced base MOCs. Some of these abstract base MOCs are depicted on the higher levels of figure 7 and reflect the characteristics of distributed applications specifically w.r.t. configuration and fault management.

The upper left part of the figure contains the classes derived from the ODP computational language; they help to model bindings and interactions between clients and servers. The upper middle part depicts a (small) section of the CIM application schema while the upper right part contains some classes from the CIM system schema. As described in section 5.1, the base MOCs do not contain application specifics. In

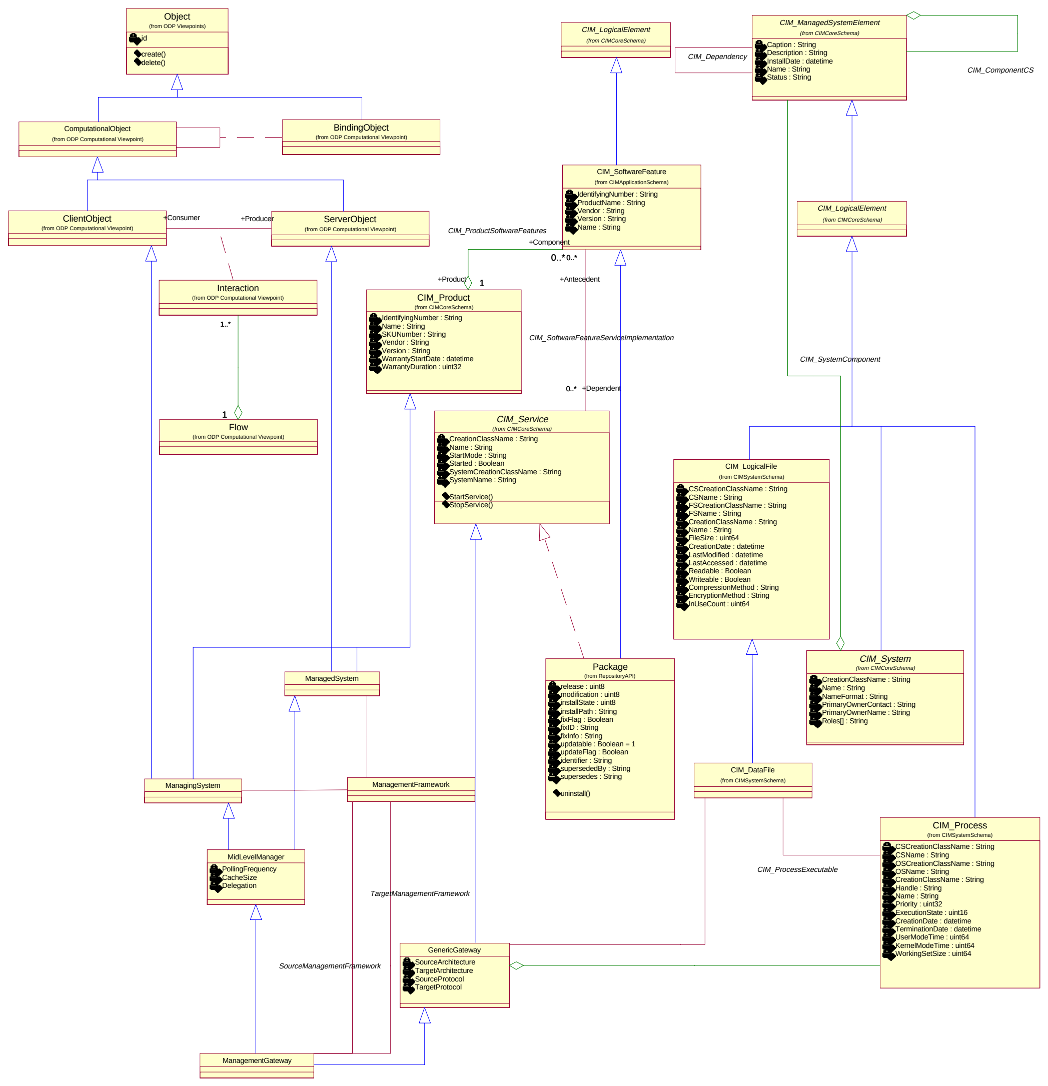


Figure 7: CIM– and RM-ODP–based object model of management systems (extract)

order to support the management of management systems and gateways, they have to be refined to more application-specific MOCs: The lower part of the figure depicts some of the classes that we have defined and which serve as the basis of our implementation.

For the sake of clarity, the following kinds of management information have been omitted from the figure: Object classes for the maintenance and control of event and alarm logs, instrumentation for traces and definitions of value constraints like thresholds. The definitions of the attributes (data types, default value, hints whether they are read-only, CIM qualifiers) and method signatures (arguments, return type) have also been suppressed. In addition, the representation uses multiple inheritance while our initial model avoids multiple inheritance by making use of association classes. However, these tend to impair the readability of the model. Furthermore, the "realizes" associations of "ManagingSystem" and "ManagedSystem" to "Package" and a couple of other self-explaining associations habe been left out in order to increase readability.

It can be seen that MOCs for "ManagingSystem" and "ManagedSystem" are derived from the ODP computational language classes “client” and “server”, respectively; this is due to the fact that management systems usually act in a client and managed systems act in a server role. The model also reflects the fact that MLMs are dual role entities and consequently inherit properties and operations from both of the above MOCs. Management gateways are – as described in section 2.1 – a special case of MLMs that additionally perform transformation activities. These transformations are determined by two issues: The first ties a Management Gateway to source and target information models (associations “SourceManagementFramework” and “TargetManagementFramework” with “ManagementFramework”); the second allows “ManagementGateway” to inherit its protocol translation capabilities from “GenericGateway”.

5.3 Prototype Implementation

For demonstration purposes, we have instrumented a commercial network management platform (*IBM NetView for AIX* with *TMN Support Facility*) and a CMIP/SNMP Management Gateway developed by us [28] with CORBA-based management agents implemented in Java. The left part of Figure 8 depicts a CORBA/Java-based management environment: CORBA agents (here: for DNS and WWW servers and AIX SMIT) are remotely administered by means of a CORBA-enabled WWW browser. CORBAservices such as Events, Notification, Topology, Managed Sets and Policy (see e.g. [46]) could provide the required management functionality in a location-independent way. However, only a minor part of these important management services have been implemented by now. Fortunately, current management platforms (right part of Figure 8) contain components that have a very similar functionality to the CORBAservices mentioned above. We therefore decided to expose the main platform components (Event Dispatcher, Topology Manager, State and Performance Monitors) to CORBA and use them as a *temporary replacement* for currently specified, but not yet available CORBAservices. We have encapsulated the APIs of the Management System with IDL wrappers (dark grey shaded arcs with black dots in Figure 8), which gives us the opportunity of reusing a large part of the Management System functionality (e.g., event filtering, topology). By doing so, we create a conceptually integrated management information base on the platform where management-related information is collected and evaluated independent of the originating base architecture. This multiarchitectural manager is able to access managed objects in OSI/TMN, SNMP and CORBA environments.

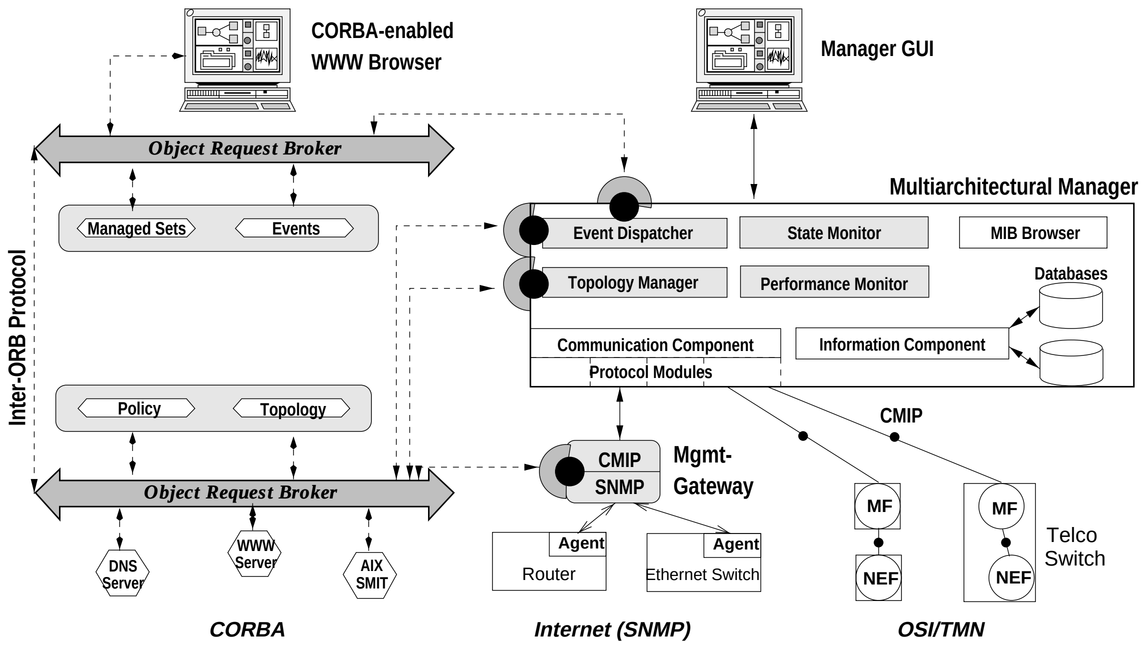
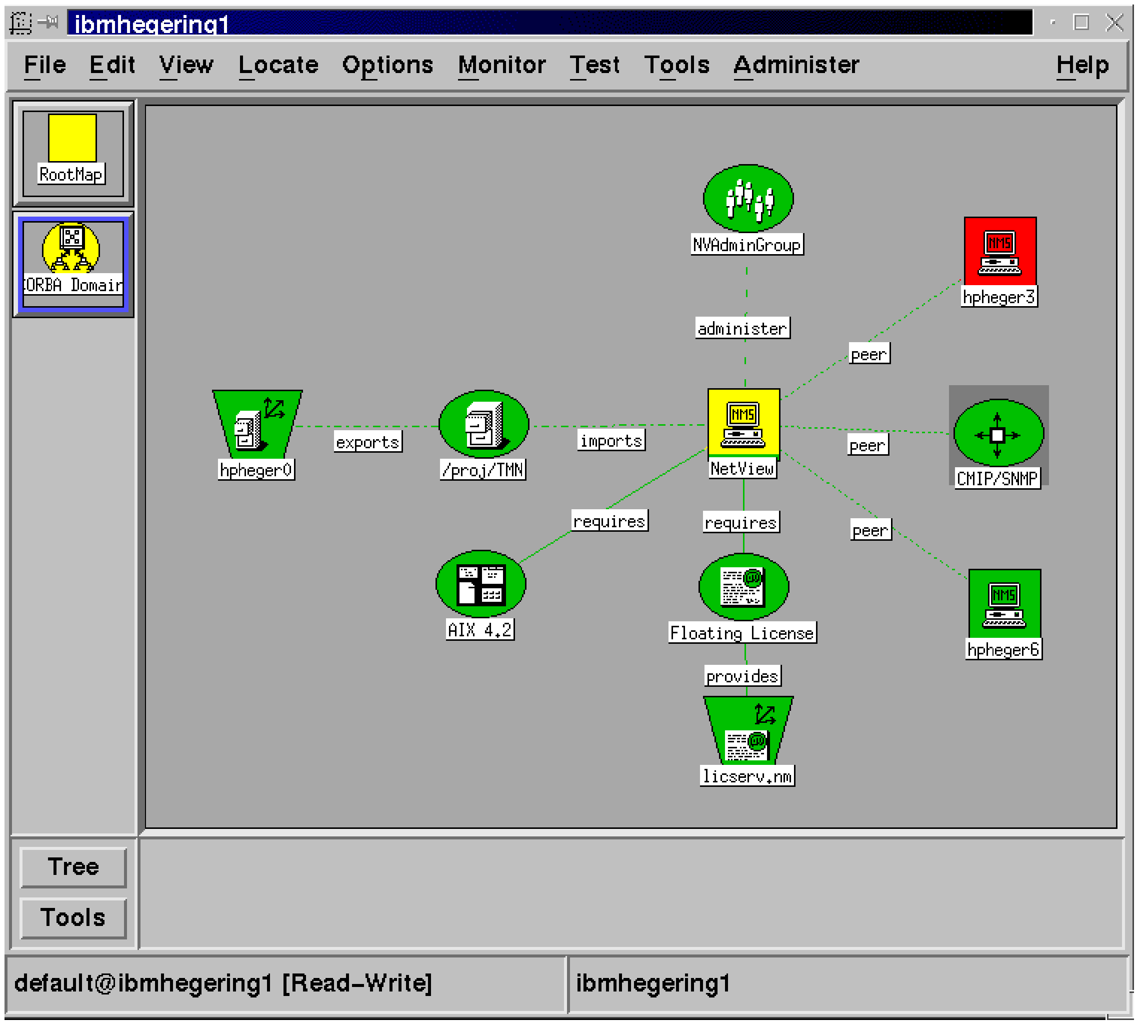


Figure 8: CORBA-based instrumentation of management systems and gateways

In addition, the IDL wrappers also provide the management instrumentation of the management system and the management gateway via CORBA. Furthermore, the representation of AIX SMIT as CORBA objects gives us access to the systems management information described in section 4.1. Together, they represent the managed objects identified in the previous sections. We are then able to manage both the management system and the management gateway via CORBA.

As we wanted to manage these systems from a web-based interface, we implemented the management application prototype as Java applets and used a commercial Java-ORB, *VisiBroker for Java*. VisiBroker is also part of the *Netscape Communicator* web browser; we therefore expected that the application would load within less than a second due to its modular design and the small size of the resulting Java classes (less than 10 kbyte per class). Unfortunately, Netscape Communicator contains only an earlier version of the Java-ORB. Consequently, the Java classes for the ORB itself and its bundled CORBAservices (in total: a Java archive of 2.5 MB code size) always have to be loaded once per session into the browser first *before* any applet can be started.

On the other hand, the access to IBM NetView is only possible through C Application Programming Interfaces. We therefore built wrappers based on the Java Native Interface around the APIs in order to



default@ibmhegering1 [Read-Write]

ibmhegering1

Figure 9: The management system’s relationships, as seen from a management platform console

access them from Java. JNI has been developed for that purpose, thus enabling programs running under the control of the Java Virtual Machine to access other programs or libraries that have been written in C. The reason for choosing Java instead of C++ stems from the fact that we intend to copy parts of the management instrumentation between different management systems at runtime.

Figure 9 shows a screendump of the prototype implementation as it can be seen from the GUI of the management console: The management system being administered (“NetView”) mounts a directory from another workstation (left part of the figure); is involved in peer relationships with two workstations and a CMIP/SNMP gateway, and has AIX 4.2 as prerequisite. In addition, it requires a floating license that is provided by the license server (bottom of the figure). The different colours of the icons provide information regarding the status of the managed objects; in addition, the logical links between the different icons are able to indicate their status by being coloured appropriately.

6 Conclusions and Areas of further Research

In this paper, a CORBA-based implementation of a prototype for managing management systems and gateways is presented. The conceptual modeling work is based on CIM and RM-ODP. The need for managing management systems arises because management systems of different authorities have to seamlessly interoperate with each other, even if they are based upon different management architectures; thus, we have analyzed and evaluated three mechanisms for achieving interoperability. On the other hand, management systems are also very complex; it is thus essential to provide appropriate management information and services. Our aim was to define such an open, common set of management information that should be applicable to a wide range of management systems and management gateways. We have developed management models for these entities by analyzing what kinds of management information and services are needed to enable an integrated management of these systems.

Key to our approach is the perception of management as a distributed application that itself needs to be managed. We are then able to apply standardized frameworks for distributed applications (such as CIM and RM-ODP) for the definition of generic MOCs that already contain a large set of management instrumentation at the top of the inheritance hierarchy. This management model is then refined into more specific object classes that reflect the structure and functionality of management platforms available on the marketplace. On the other hand, we can infer that a systematic and structured solution for the problem of managing management systems contributes also to solving the problem of application management. We have implemented the object model described in this paper for instrumenting the network management platform *NetView for AIX* by Java/CORBA-based management agents so that it can be managed from a web-based interface. Apart from determining dependency relationships between instantiated components, almost the complete set of required management information can be provided by this management system. However, this information is scattered across different sources such as platform APIs, configuration files and configuration tools of the underlying operating system. A large part of our implementation work consisted in encapsulating all this information under a single type of interface.

We have experienced that the association of “ease of use” with the term “web-based management” may apply to the actual system user, but definitely not for the developer: Several hundred object classes tied together by various kinds of associations require a deep knowledge of object-oriented design methodologies such as UML. “Wallpaper-size” diagrams for every CIM schema containing exhaustive inheritance hierarchies underline the fact that CIM inhibits a considerable degree of complexity. Despite the large amount of classes, CIM doesn’t provide yet concepts comparable to the RM-ODP computational language; therefore, we had to add the appropriate object classes from RM-ODP to our CIM-based model in order to obtain the desired functionality. However, it is expected that the work of the DMTF Application Management Group on CIM runtime extensions will make such substitutions obsolete.

References

[1] Application Management Specification. Version 2.0, Tivoli Systems, November 1997.

[2] Application Component Team. Generic Requirements for Telecommunications Management Building Blocks. Part 1 of the Technology Integration Map, GB909 (Part 1) v. 2.04 draft, TeleManagement Forum, June 2000.

[3] Application Response Management. Version 2.0, Tivoli Systems, November 1997.

[4] M. Bauer, R. Bunt, et al. Services Supporting Management of Distributed Applications and Services. *IBM Systems Journal*, 36(4):508 – 526, 1997.

[5] L. Bennett. *Multiprotocol Network Management: A Practical Guide to NetView for AIX*. McGraw-Hill, 1996.

[6] P. Bhoj, S. Singhal, and S. Chutani. SLA Management in Federated Environments. In Sloman et al. [38], pages 293–308.

[7] P. Brusil, J. Hellerstein, and H. Lutfiyya. Applications Management – Current Practices, Research Results, and Future Directions. *Journal of Network and Systems Management*, 6(3):361 – 366, September 1998.

[8] W. Bumpus, J.W. Sweitzer, P. Thompson, A.R. Westerinen, and R.C. Williams. *Common Information Model: Implementing the Object Model for Enterprise Management*. J. Wiley & Sons, 2000.

[9] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999.

[10] Middleware: The Glue for Modern Applications. *Gartner Advisory, Strategic Analysis Report*, July 1999.

[11] I. G. Ghetie. *Network and Systems Management: Platform Analysis and Evaluation*. Kluwer Academic Publishers, 1997.

[12] German Goldszmidt and Yechiam Yemini. Delegated agents for network management. *IEEE Communications Magazine*, 36(3):66–70, March 1998.

[13] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems — Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, 1999.

[14] L. Lawrence Ho. Interoperable Distributed Management. *Journal of Network and Systems Management*, 7(1):137 – 140, March 1999.

[15] J. Hong, M. Katchabaw, A. Bauer, and H. Lutfiyya. Distributed Applications Management Using the OSI Management Framework. Technical report, TR-448, University of Western Ontario, Department of Computer Science, January 1995.

[16] IBM Corporation. *AIX Version 4.3 General Programming Concepts: Writing and Debug ging Programs*, October 1997. Chapter 17: Object Data Manager (ODM).

[17] IBM Corporation, International Technical Support Organization, Research Triangle Park, NC 27709-2195. *Designing Tivoli Solutions for End-to-End Systems and Service Management*, June 1999. Order Number: SG24-5104-00.

[18] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - System Administration - Part 2: Software Administration. IEEE Standard 1387.2, The Institute of Electrical and Electronics Engineers, 1995.

[19] Open Distributed Processing – Reference Model. IS 10746, International Organization for Standardization and International Electrotechnical Committee, 1995.

[20] Open Distributed Processing – Reference Model – Part 3: Architecture. IS 10746-3, International Organization for Standardization and International Electrotechnical Committee, 1995.

[21] Inter-Domain Management: Specification Translation. Preliminary Specification P509, The Open Group, March 1997.

[22] M. W. Johnson and S. Smead. *Beyond ARM 2.0 - API Extensions that enable pervasive Service Level Instrumentation*. Computer Measurement Working Group, December 1998.

[23] C. Kalbfleisch, C. Krupczak, R. Presuhn, and J. Saperia. Application Management MIB. RFC 2564, IETF, May 1999.

[24] Pramod Kalyanasundaram and Adarshpal Sethi. Interoperability Issues in Heterogeneous Network Management. *Journal of Network and Systems Management*, 2(2):169 – 193, June 1994.

[25] G. Kar, A. Keller, and S.B. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In Douglas Zuckerman, editor, *Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium*, pages 61–75. IEEE Press, April 2000.

[26] M. Katchabaw, S. Howard, H. Lutfiyya, and A. Marshall. Making Distributed Applications Manageable through Instrumentation. *The Journal of Systems and Software*, (45), 1999.

[27] A. Keller. *CORBA-basiertes Enterprise Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung*. PhD thesis, Technische Universität München, October 1998. (*in German*).

[28] A. Keller. Tool-based Implementation of a Q-Adapter Function for the seamless Integration of SNMP-managed Devices in TMN. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 98)*, pages 400–411, New Orleans, USA, February 1998. IEEE Press.

[29] A. Keller and B. Neumair. Using ODP as a Framework for CORBA-based Distributed Applications Management. In J. Rolia, J. Slonim, and J. Botsford, editors, *Proceedings of the Joint International Conference on Open Distributed Processing (ICODP) and Distributed Platforms (ICDP)*, pages 110–121, Toronto, Canada, May 1997. Chapman & Hall.

[30] C. Krupczak and J. Saperia. Definitions of System-Level Managed Objects for Applications . RFC 2287, IETF, February 1998.

[31] S. Mazumdar, S. Brady, and D. Levine. Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries. In *Proceedings of the 3rd IFIP/IEEE International Symposium on Integrated Network Management*. North-Holland, April 1993.

[32] Subrata Mazumdar. Inter-Domain Management between CORBA and SNMP. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, L'Aquila, Italy, October 1996.

[33] T. Müller. CORBA-basiertes Management von UNIX-Workstations mit Hilfe von ODP-Konzepten. Master's thesis, Technische Universität München, February 1998. (*in German*).

[34] G. Pavlou. A Novel Approach for Mapping the OSI-SM/TMN Model to ODP/OMG CORBA. In Sloman et al. [38], pages 67–82.

[35] Rational Software Corporation. *Rational Rose 98i: Using Rational Rose*, February 1998.

[36] A. Schade and P. Trommler. A CORBA-based Model for Distributed Applications Management. In *Proceedings of the IFIP/IEEE Seventh International Workshop on Distributed Systems: Operations & Management (DSOM'96), L'Aquila, Italy*. IEEE, October 1996.

[37] A. Schade, P. Trommler, and M. Kaiserswerth. Object Instrumentation for Distributed Applications Management. In *Proceedings of the IFIP/IEEE International Conference on Distributed Platforms (ICDP'96), Dresden, Germany*. IFIP, Chapman and Hall, February 1996.

[38] Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors. *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, Boston, MA, USA, May 1999. IEEE Publishing.

[39] Nader Soukouti and Ulf Hollberg. Joint Inter-Domain Management: CORBA, CMIP and SNMP. In A. A. Lazar and R. Saracco, editors, *Proceedings of the 5th International IFIP/IEEE Symposium on Integrated Management (IM)*, pages 153–164, San Diego, USA, May 1997.

[40] R. Sturm and W. Bumpus. *Foundations of Application Management*. J. Wiley & Sons, 1998.

[41] Application Component Team. Requirements for Management of ORB-based Telecommunications Management Building Blocks. ACT Working Document ACT 01-99, TeleManagement Forum, April 1999.

[42] OMG Unified Modeling Language Specification. Version 1.3 ad/99-06-08, Object Management Group, June 1999.

[43] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.

[44] Chris Wellens and Karl Auerbach. Towards Useful Management. *The Simple Times*, 4(3):1–6, July 1996.

[45] Systems Management: Distributed Software Administration. CAE Specification C701, The Open Group, January 1998.

[46] Systems Management: Common Management Facilities Volume 1. Preliminary Specification P421, X/Open Ltd., June 1995.

[47] UNIX and NT Performance Management: A Discussion of the Primary Issues. *Yankee Group Report, Management Strategies*, 8(9), May 1998.