21

Flexible Management of ANSAware Applications*

B. Meyer[a] and C. Popien

[a]Aachen University of Technology, Computer Science IV,
Ahornstr. 55, 52056 Aachen, Germany, e-mail: bernd@i4.informatik.rwth-aachen.de

In contrast to monolithic application programms developed for large mainframe computers, applications for networks of interconnected workstations are nowadays developed using a client-server model of computer cooperation. Applications are still monolithic, but using certain servers for subtasks. The next step will be to devide an application into smaller subparts with common service and resource requirements in order to build units for concurrency and migration. These components of an application also need to be managed. We present a new approach to application management. For the distributed computing platforms ANSAware we developed monitoring and analysis tools. In addition we developed a formal notation for defining policies. Our approach to application management introduces policy controller objects, that enforces policies for policy target objects. Experiences gained with policy descriptions lead to a common policy handling function as it is required for Open Distributed Processing and management systems.

Keyword Codes: C.2.4; D.2.9;
Keywords: Distributed Systems, Management

1. INTRODUCTION

Applications running on large mainframe computers have a monolithic structure with respect to their resource utilisation. Ongoing with the evolution of computer and communication technology, modern computer systems are able to share their resources over different kinds computer networks. This enables applications to be distributed over a number of computer nodes with an expected performance gain. In order to cope with problems arising from distribution, a number of computing platforms have been developed, e.g. the Distributed Computing Environment (DCE) of the Open Software Foundation [DCE], the ANSAware [ANSA] and implementations of the Common Object Request Broker Archictecture (CORBA), that is a part of the Object Management Architecture (OMA) of the Object Management Group [OMG]. In addition, the ISO advanced standardization work on Open

* This work was partially supported by the Deutsche Forschungsgemeinschaft under Grant No. Sp 230/8-1

K. Raymond et al. (eds.), *Open Distributed Processing*

Distributed Processing (ODP) for defining a framework for classification, development and interworking of distributed processing platforms, see [ODP p1-p3]

Besides proceeding appropriate functions to support distributed processing, the managament of distributed systems becomes increasingly important. Most management concepts and systems have been developed for management of network components. In the last years a lot of work has been done in adopting these concepts for managing resources of a distributed system. Because of new requirements put up, network management concepts need to be extended. The notion of a service has been introducted as a further abstraction from underlying processing and networking resources. These services offer more functionality as those offered by an (distributed) operating system or a communication network. However, this distinction is a more logical one and is a bit fuzzy, e.g. mailing or database services are more complex services than a transport layer service or a file service. In this work are thinking of services in the former sense. Service management became an important area of research with the rising popularity of the client-server model for computer cooperation, see also [PK94], [PKM94]. In addition to services offered to a wide range of users, an application contains a huge amount of code specific to that application. Consistently, managing applications in a distributed system is a more complex task than service management. On the other side, application management has to fall back on service and resource management in order to not invent the same concepts again.

Currently distributed systems are becoming more and more large, containing hundreds of computer nodes. The configuration of these systems is likely to change dynamically due to a node or server failure. In addition, new services or components are introduced or removed from the system. To cope with changing configurations, components of distributed systems need to have flexible behaviour to react on changes. This behaviour has to be managed by human or computer manager agents. A reasonable approach to this goal have been introducted by so-called policies. There has been a lot of research on the topic of policies, e.g. see [MoSl94], [Wie94].

The paper is structured as follows: the second section gives an introduction to basic management concepts and tasks of application management. A policy-based approach to application management is introduced in section three. Therefore a model for describing applications and policies is given. Section four shows, how these models can be realized using the ANSAware distributed platform as a base for an implementation of application management. Finally fields for future work will conclude this work.

2. MANAGEMENT OF APPLICATIONS IN DISTRIBUTED SYSTEMS

In the following, we introduce a number of notions and their interpretation according to the reference model of Open Distributed Processing (ODP), see [ODP p2]. Domains consists of a collection of managed objects, that are grouped together for a special purpose. One domain is managed by a single controller object. Policies can be defined an relation between one or more objects and some reference behaviour. The behaviour can be expressed in terms of obligations, permissions and prohibitions. An obligation behaviour must be fulfilled,

whereas permitted behaviour is allowed to occure. A prohibition is in some way the opposed behaviour to an obligation, because it defines a behaviour, that must not occur.

Management concepts and tools can be classified using three orthogonal dimensions [HeAb93], see figure 2.1 . One dimension refers to the functional management areas as defined by OSI Management, namely accounting, configuration, fault, perfomance and security management [OSI Man]. Another is called time dimension and deals with phases in the life of a management application or a tool the description aims at. Some tools are only suited for planning an application, wheres others focus on its development or maintainace. The last dimension deals with the target object a management tool works on. There exists a large number of tools for network and system components. But also applications and enterprises need to be managed.

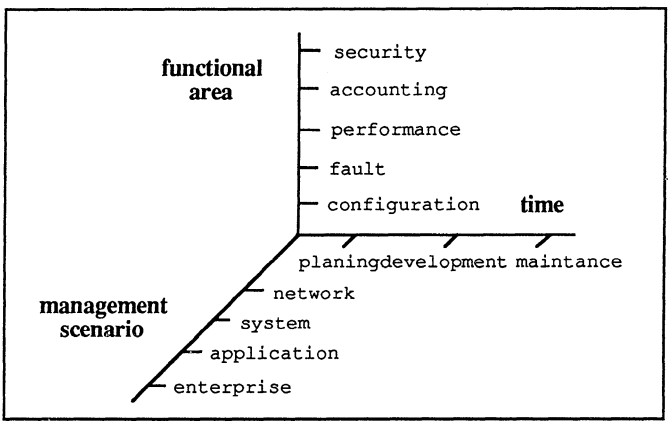

Figure 2.1. Three dimension of management

Management in ODP defines five different functional areas, namely quality of service, accounting and configuration management accompanied by monitoring and defining policies [Slo90], [ODP p1]. QoS Management encompasses fault and performance management, whereas security is classified to be an operational and not a managament function, same as e.g. communication.

Target of application management are components like applications, object cluster, application objects, server objects, interfaces and binding objects . According to the functional management areas, application management requires:

accounting management

- administration of payment methode for service usage
- administration of cost calculation methode for service usage

configuration management

- controlling the state of an application component
- storing properties of application components

- starting and stopping of application components
- modifying general configuration
- administration the migration of application components

fault management

- administration of application components checkpoints
- fault tolerant service usage
- collecting fault messages
- recognition of cause of fault messages

performance management

- monitoring performance characteristics of an interface, e.g. number of operation , invokations, volume of transferred data, response time
- monitoring performance characteristics of an application component, e.g. throughput, total delay, average service time
- determining performance bottlenecks
- avoiding performance bottlenecks
- monitoring and controlling reservation of network and system resources
- load balancing of application components

security management

- administration of authentication for multiple server access

3. POLICY-BASED APPLICATION MANAGEMENT

The following section introduces a policy-based approach to application management. First a model for applications and policies are presented, which are used by application management. Using these models a concept for an integrated application management tool will be developed.

3.1 The application model

In our approach an application consists of a number of cooperating objects. Each object is able to interact with other objects via operations it offers at its interfaces. In general, objects can have more than one interface; e.g. for management purposes there will be a separate management or control interface. The interfaces hides the internal object state and can only be manipulated by invoking interface operations or it forwards notifcations initiated by its internal objetc state fulfilling a given condition. Binding between objects can be established by an binding object, if it has to be managed by the application itself or a management application. A binding object is an abstraction from the communication channel established to enable interworking of two (or more) objects. The binding control interface enables users to change properties of a binding, e.g. quality of service parameter. New cooperating partners can be introduced or connected ones can be withdrawn. Applications of these capabilities are data stream handling as it is necessary for multimedia systems or control systems for technical production processes. Application objects with common service and resource requirements can be grouped into object clusters. Clustering is a means of configuration and an object

cluster is intendend to be located at the same node as its resources or servers. In general not all functionality is provided by application objects, but a number of (external) server objects are also needed. These servers offer different services, e.g. operating system services like a file system, memory management, persistent data storage service offered by a database systems or a global name service according to the X.500 recommendations. All introduced application components can be mapped on ODP modelling concepts.

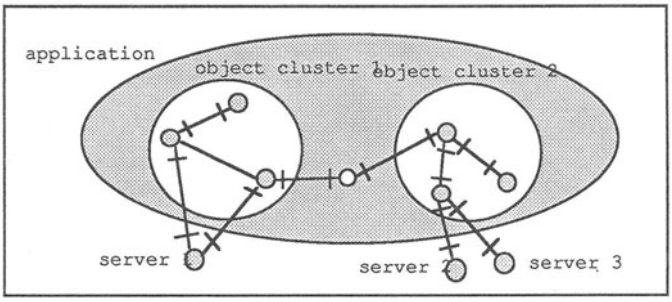

Figure 3.1. Components of the application model

In the following, we will give examples of information required by application management for each component of an application according to our model. These metrics can be useful either as current, or average value. For certain analysis tasks, a complete value history is required.

Table 3.1. Application management metrics

component	metric
application	required services (resources)
	contained object cluster / application objects
	application performance metrics (waiting time, throughput)
object cluster	required services (resources)
	used server objects
	contained application objects
	host computer node
	cluster performance metrics (response time, throughput)
application object	required services (resources)
	used server objects
	host computer node
	cluster performance metrics (response time, throughput)
server object	number of current users / interfaces
	waiting queue length
	server performance metric

component	metric
interface	number of invokations per operation
	number of failed operation invokations
binding object	protocol type
	communication channel peformance metrics (response time, throughput)
	bit error rate

3.2 The policy model

As mentioned above, policies in ODP are defined as an prescriptive relation between one or more objects and some reference behaviour. The behaviour can be expressed in terms of obligations, permissions and prohibitions. An obligation behaviour must be fulfilled, whereas permitted behaviour is allowed to occure. A prohibition is in some way the opposed behaviour to an obligation, because it defines a behaviour, that must not occur.

Our policy model has a certain view on objects it deals with. Objects are assumed to be black boxes hiding their internal state behind an interface. The policy state of an object is determined by the kinds of policy related to it and the current behaviour associated with each policy. In our model policy changes are caused by intervenition of a (human) system operator or by the management system itself. Policies in the presented approach are responsible for extending, reducing or modifying the behaviour of certain objects. It defines conditions under which a change of policy state appears, but it does not completely specify the object behaviour. Conditions for an automatic change can be external events like the reception of a notification or the occurance of an internal state. At a higher level of abstraction, policies only deal with event, behaviour or activity identifiers. They can be further refined by manipulating interfaces of objects. In the case of application management these objects are all above defined application components. For example, a certain behaviour can be achieved by restricting an operation parameter to a special value. In other cases, an activity defined in a policy definition correspondes to a certain operation (or a number of operations) at an interface. Prohibiting this activity can be done by disabling these operations at the corresponding interface. In addition it is possible, that the administrator role of enforcing a policy can be delegated from one object to another. An important design decision to be made for a policy-based management system is the question whether a policy should be handled at run-time or at compile-time. It is well-kown that interpreters are more flexible, but compiled code shows better performance. Due to promise of focussing on flexibility and scalability, an interpreter approach is chosen in this approach. Testing conformance of a given object against a behaviour determined in a policy is a very important task, but it is out of scope in our approach yet. Therefore, existing formal description techniques like SDL, LOTOS or Estelle, that allow conformance testing should be investigated for their usage as behaviour description for policy behaviour, see also [Tur93]. In our approach, it is assumed, that all objects are able to show the behaviour requested by a policy. Otherwise, a policy failure notification is to be forwarded. One difficulty in using existing techniques are the policy modalities, that are not

directly expressable Modalities can be expressed in terms of model logic or temporal expressions over an state-transition model, see [HeMi85], [Got92]. For example, a forced behaviour can be interpreted by the model operator always. Therefore a state-transition model is constructed by policy states and changes of policy as transitions. It must be checked, that the forced behaviour is enabled in every state of the state-transition model. This procedure is called model checking and is a seperate field of research. Policies are a general purpose concept required for managing distributed systems. Their usefullness has been recognized in several projects and finally standarization is concerned about a general policy handling function. Although it is a typical management function, not only OSI Management tries to include policies, but also ODP is using policies for startegic modelling. In our point of view, a policy handling function will need to include at least activities like

- storing, naming and retrieving a policy,
- replacing or modifying a policy,
- merging two policies,
- solving conflicts between two policies or
- applying well-defined refinements on a policy.

3.3 Integrated model for application management

The environment of a policy-based application management system is shown in figure 3.2. Besides other objects, the application management contains a policy manager. Application components like whole applications, object cluster, objects, interfaces, binding objects and server objects are managed by the management system.

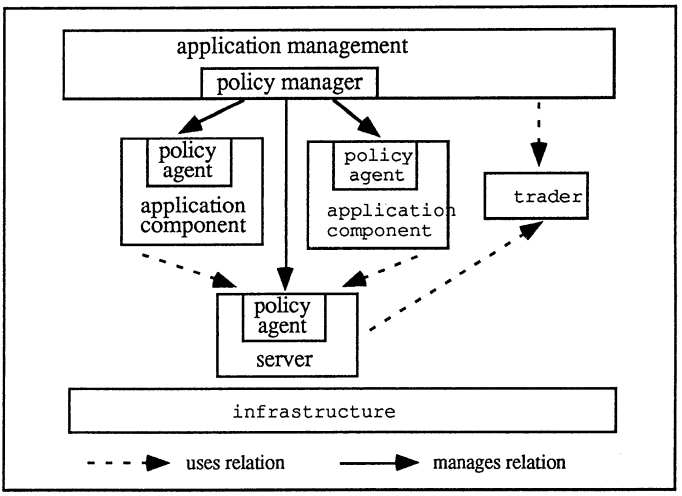

Figure 3.2. Global architecture of an application management system

Each application component and service object is equipped with a policy agent, which is in charge of controlling object behaviour with respect to its policy. This policy agent itself is part of a management agent, that is in charge of additional management task. For simplicity, the management agent is not integrated into figure 3.2. Of special importance in our application management model is the trader component [ODP TF]. It is used by the application management to get server interface references for services required by an application. In our approach it serves as a means to achieve a higher level of software dependability. Especially if a server failure occurs at run-time, it is necessary, that the application management is able to request a new server interface and to restart the failed operation call. Therefore it must be assured, that modifications already done by the failed server have to be cancled.

3.4 Related work

The META approach presented in [MCW+91] also uses policies to achieve flexible management behaviour. It also makes use of policies, but the only statement possible is a conditioned activity. Distributed applications are monitored by so-called sensors and operations on applications can be invoked by so-called actuators. Together, both concepts offer similar functionality that is given by the managed object concept defined in OSI Management. Another approach to distributed applications management has been developed at the universtity of Frankfurt within a project names PRISMA (a platform for integrated construction and management of distributed applications), see [ZBD+94], [ZFD93]. It supports an application model consisting of interfaces, communication contexts, application components and a global application configuration. Interfaces and components are related to either the application or the management. A new specification language has been introduced, but in addition these concepts are described using the guidelines for use of managed objects (GDMO). A related approach is established by the MELODY (management environment for large open distributed systems) project of the universtity of Stuttgart [KoWi94], [Kov93]. There an activity management has been proposed, which can be compared to transactions in database systems; except a complete rollback. An activity consists of a sequnce of service calls, where the actvity state is checkpointed after each service operation. On service failure, the activity is set to the last checkpoint and a new server is imported using a trader. The service call is then invoked again. The application model within MELODY assumes interacting components to as building blocks. A component itself is characterized by its type, its interfaces and the role it can play in an interaction.

4. IMPLEMENTATION USING ANSAWARE

ANSAware is an infrastructure for developing and running distributed applications. It is available for a number of operating systems like SunOS, HP/UX, VMS and MS DOS. Communication between computer nodes must be provided by a socket interface of a transport service using the TCP or UDP protocol. Besides services like remote procedure call (RPC) and thread support, ANSAware provides three server. One server is called the factory and is able to start server objects of a certain service type. An interface reference has to be

obtained, before a client can use an object service. This reference can be obtained by making an service import request to the trader. The trader is the second server provided by ANSAware and delivers interfaces references of servers of a given service type. Before interface references to server objects can be forwarded, they have to be exported to the trader. In addition, cooperations between different trader server can be established.

Application management is based on management-relevant information about application components, binding and server objects. A means to this end is a monitor tool for distributed systems. We developed such a tool for ANSAware called ANSAmon [Hei94]. A special kind of these management information, performance metrics, can be calculated by analysing or simulating application models. They fall back on system parameters measured by a monitoring system. Another way to achieve information about performance characteristics is to use analytical techniques. Therefore we developed a method to evaluate fork-join queueing networks, see [MePo93], [PMS94].

In order to make our policy model understandable to computer, we developed a formal notation for policy description called the policy definition notation (PDN), see [MePo94]. It is based on the above defined object model. Here only a brief summary of the most important constructs will be given. Policies using PDN are named and consist of descritpion of a domain it is applied to and a behaviour description. The domain policy is enforced by a special controller object. Policy changes are initiated by the occurance of external events or internal state conditions. The prescribed policy behaviour can be an obligation, a permission or a prohibition.

<policy>::= "**policy**" <name> "**for**" <domain> "**with**" <behaviour_desc> "**end policy**".
<behaviour_change> ::= <event_triggered_behaviour> | <conditional_behaviour> .
<event_triggered_behaviour> ::= "**on**" <external_event> "=>" <behaviour>.
<conditioned_behaviour> ::= "**if**" <internal_state> "**then**" <behaviour>.
<policy_behaviour> ::= <modality> <behaviour>.
<modality> ::= "**force**" | "**permit**" | "**prohibit**" | ... | <empty>.

Figure 4.1 describes a policy for a trader domain, that is managed by a trader administrator. It determines where to place a new user depending on the size of the trader database (TDB) of each trader. It is assumed, that a user contacts its local trader first. If a database is changed, the corresponding trader sends a noticication to its administrator.

Policies can be generated by system administrators or directly by a management application. Refinement of policies likelywise needs to be done by human users. There is some research on developing automatic refinement tools, but we are convinced, that a certain amount of human activity will still remain. Changing current policy behaviour of an object can be initiated by a system administrator or by an analysis object. Analysis needs values for system and application parameters to be gathered by a monitoring system. Policies created or modified in the policy maker are defined using our notation and will be interpreted by a policy agent. It controls the behaviour of the distributed application and in some cases of the distributed computing environment, too.

```
policy UserPlacementPolicy
for TRADER enforced by trader_administrator with
behaviour AdaptivePlacement for trader_administrator is
    force
        on notification reception TDB_Updated from TRADER with TDB_Size =>
            if (SizeTDB in [0, 320) )
                then behaviour LocalPlacement for TRADER
            if (SizeTDB in [320, 550) )
                then behaviour PlaceAtLightestLoadTrader for TRADER
            if (SizeTDB in [550, 600) )
                then behaviour LocalPlacement for TRADER
where
    behaviour LocalPlacement is ...
    behaviour PlaceAtLightestLoadTrader is ...
end policy
```

Figure 4.1. Example policy for flexible placement of trader user requests

ANSA objects to be managed by policies are supported by an policy controller. This controller manages the access to the server object. Figure 4.2 shows a policy manageable trader object. Internally it consist of a policy controller and a server object. The trader has a management interface and a separate service interface. A system administrator or an management application can change policy via the managemebt interface, whereas the service interface is the same as the server interface, but interaction is controlled by the policy controller.

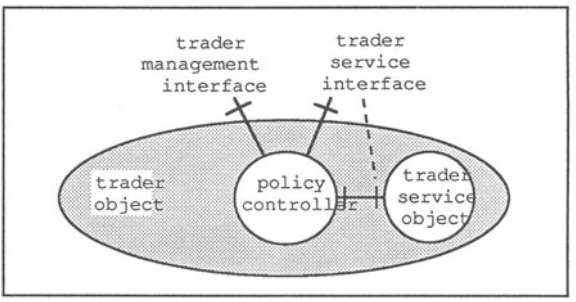

Figure 4.2. Policy-controlled trader object

There might be other contributing to the trader management interface, but for convinience we chose only one policy controller. Common operations at management interfaces of objects can devided into operations on one policy description and on the list of available policy descriptions. There are operations to add a new policy to a list, delete a certain policy from a list and to show all policies with the list. Policies can be read and modified by a new policy description. In addition, several notifications can be forwarded from the policy controller, e.g.

a notification announcing a policy behaviour change due to an external event or an internal condition.

5. CONCLUSIONS

In this paper we presented a policy-based approach to application management in distributed systems. Policies enable the application behaviour to be managed automatically. It consists of an application component model, that is based on concepts introduced by the RM ODP. Besides we developed an appropriate policy model. For the distributed computing environment ANSAware we present an application management approach. It makes use of a distributed montioring tool we developed. Policies are described formally using a new notation, which allows to be interpreted by computer. We propose to add an policy controller object to each object, whose behaviour is determined by a policy. This controller is able to enforce a policy by influencing interface activities. The policy controller object offers behaviour required by a general policy handling function. Future work will be done in the area of performance, fault and configuration management applications.

REFERENCES

[ANSA] ANSAware Release 4.1, Manual Set, Document RM.102.02, February 1993

[DCE] Open Software Foundation: Introduction to OSF DCE. Prentice Hall 1992

[Got92] Gotzhein, R.: Temporal Logic and applications - a tutorial. Computer Networks and ISDN Systems, Vol. 24, 1992, pp. 203-218

[HeAb93] Hegering, H.; Abeck, S. : Integrated Network- and Systemsmanagement (in german). Addison Wesley, 1993

[Hei94] Heineken, M.: Performance Analysis of the Communication Components of the ODP Prototype ANSAware (in german). Diploma Thesis at Aachen University of Technology, Nobember 1994

[HeMi85] Hennessey, M.; Milner, R.: Algebraic Laws for Non-determinism and Concurrency. Journal of the ACM, Vol. 32, 1985, pp. 137-161

[Kov93] Kovács, E.: The MELODY management system for distributed applications (in german). Proceedings of EMVS´93, Frankfurt, 1993

[KoWi94] Kovács, E.; Wirag, S.: Trading and Distributed Application Management: An Integrated Approach. Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management, Toulouse, October 1994

[MCW+91] Marzullo, K.; Cooper, R.; Wood, M. et al: Tools for Distributed Application Management. IEEE Computer, August 1991, pp. 42-51

[MePo94] Meyer, B.; Popien, C.: Defining Policies for Performance Management in Open Distributed Systems. Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management, Toulouse, October 1994

[MoSl93] Moffet, J.; Sloman, M.: Policy Hierachies for Distributed Systems Management. IEEE Journal on Selected Areas in Communications, Vol. 11, No. 9, December 1993, pp. 1404-1414

[ODP p1] ISO/IEC JTC1/SC21/N. (7S023) Information Technology - Open Distributed Processing - Basic Reference Model - Part 1: Overview. Meeting output for Committe Draft, July 1994

[ODP p2] ISO/IEC JTC1/SC21 N7988 Information Technology - Open Distributed Processing - Basic Reference Model - Part 2: Descriptive Model. Committe Draft, December 1993

[ODP p3] ISO/IEC JTC1/SC21 N8125 Information Technology - Open Distributed Processing - Basic Reference Model - Part 3: Prescriptive Model. Committe Draft, December 1993

[ODP TF] ISO/IEC JTC1/SC21 N8409 Information Technology - Open Distributed Processing - ODP Trading Function. Working Document, January 1994

[OMG] Object Management Group: The Common Object Request Broker Architecture: Architecture and Specification. Revision 1.2, December 1993

[OSI Man] ISO/IEC IS 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework. International Standard, November 1989

[PK94] Popien, C.; Kuepper, A.: A Concept for an ODP Service Management. Proceedings of IEEE/IFIP Network Operations and Management Symposium, (NOMS'94), Kissimmee/Florida 1994

[PKM94] Popien, C.; Kuepper, A.; Meyer, B.: Service Management - The Answer of new ODP Requirements. In: Meer, J. de; Mahr, B.; Storp, S.: Open Distributed Processing II (ICODP'93), Berlin 1993, North Holland 1994, pp. 408-410

[PMS94] Popien, C.; Meyer, B.; Sassenscheidt, F.: Efficient Modeling of ODP Trader Federations using P^2AM. (in german). In: Wolfinger, B. (ed.): Innovations for Computer and Communication Systems. Springer 1994, pp. 211-218

[Slo90] Sloman, M.: Management for Open Distributed Processing. Proceedings of 2nd IEEE Workshop on Future Trends of Distributed Systems, 1990

[Tur93] Turner, K. (ed.): Using Formal Description Techniques - An Introduction to Estelle, LOTOS, and SDL. Wiley 1993

[Wie94] Wies, R.: Policy Definition and Classification: Aspects, Criteria, and Examples. Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management, Toulouse, October 1994

[ZBD+94] Zimmermann, M.; Berghoff, J.; Doemel, P. et al: Integration of Managed Objects into Distributed Applications. Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management, Toulouse, October 1994

[ZFD93] Zimmermann, M.; Feldhoffer, M.; Drobnik, O.: Distributed Applications: Design and Implmentation (in german). PiK, Vol. 16, No. 2, 1993, pp. 62-69