Policy-Enabled Applications

E. Meier
F. J. Hauck

July 1999 TR-I4-99-05

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

Policy-Enabled Applications

Erich Meier, Franz J. Hauck
{meier,hauck}@informatik.uni-erlangen.de
University of Erlangen-Nürnberg
IMMD 4, Martensstraße 1
D-91058 Erlangen, Germany

Abstract. The lack of scalability of distributed systems is a severe problem when these systems have to grow to a large or even global scale. The administrative dimension of scalability is strongly influenced by the number of different organizations exerting control over parts of the distributed system. Each organization has its specific requirements and policies that have to be fulfilled by the distributed system.

In this paper we present a novel software architecture introduced as *policy-enabled applications* that allows external entities to define their own requirements and policies to influence the behavior of the distributed application. We identify four different roles that entities formulating policies play: developer, application administrator, domain administrator and user. Interactions between users and distributed applications are modelled by user requests. Unlike traditional applications, the processing of a user request by a policy-enabled application is not directly accomplished by mechanisms and protocols. The request is rather mapped onto a proper set of mechanisms and protocols by invoking a policy engine which evaluates the policy rules configured by the different roles. User requests should be as abstract as possible to leave enough degrees of freedom to the policy engine so that the probability of a successful mapping is maximized. With a high probability of successful user request processing policy-enabled applications exhibit a superior administrative scalability compared to traditionally designed ones.

1 Introduction

With an increasing interest in distributed systems, the scalability of these systems—or better the lack of it—becomes a very important issue. Distributed systems as the World Wide Web (WWW) or the Distributed Computing Environment (DCE) run into problems to meet their goals when growing to a larger or even global scale.

Scale can be separated into three dimensions: numerical, geographical and administrative [Neu94]. The numerical dimension consists of the number of objects (e.g., users, services) in the system. The geographical dimension is determined by the distance between these objects which depends of the topology of the distributed system. The administrative dimension finally is influenced by the number of separate organizations taking part in the distributed system.

The problems with the numerical and geographical dimensions of scale are commonly tackled by techniques like replication, distribution and caching. Objects are distributed all over the system, their states are replicated, and information once obtained is reused as often as possible. However, only few concepts are known how to deal with the administrative dimension of scalability.

The administration of a large-scale distributed system gains complexity with the number of organizations exerting control over parts of the system. Each organization is normally forming a domain within the distributed system. A domain can be described as a group of hosts, users and services. Identifiers for these objects (e.g., names of users and services) are normally only valid within a domain. Domains can form hierarchies with the introduction of subdomains. An example for the usage of subdomains is the creation of departments and workgroups within large enterprises. An even more important characteristic of domains is that their administrators often apply policies that have domain-wide validity.

Improving the administrative scalability of distributed systems now has two main issues: on one hand, growth over domain boundaries must not add too much complexity to the administration of the system. That means, systems must not prevent themselves from growing. On the other hand, with the lack of user and—more important—administrator acceptability within the local domain, people prevent these systems from growing to a larger scale. Many administrators think: “If I am not able to make this system compliant to my local policies, I will not let it run in my local domain.”

Enabling local domain administrators to set conditions for the functioning of the system, to make the distributed system comply to local policies, and to adapt it to the needs of their local users increases acceptability among administrators and users and therefore increases the administrative scalability of the system.

Our novel approach introduced as *policy-enabled applications* allows external entities to define their own requirements to influence the behavior of the distributed application. This is achieved by a strict separation of mechanism and policy code within the application, which allows the policy code to be influenced by these entities through the definition of policy rules. We identify four different role classes that entities formulating policy rules play: developer, application administrator, domain administrator and user. These role classes allow us to define requirements for the design of a scalable policy subsystem.

Interactions between users and distributed applications are modelled by user requests. After being formulated by the user or client, user requests are sent to the application. An example of a user request would be a WWW browser requesting a WWW proxy server to retrieve a document with a certain URL. Traditional applications perform the processing of user requests directly by using mechanisms and protocols. In our policy-enabled application model, user requests are mapped onto concrete mechanisms and protocols by invoking a policy engine which evaluates the policy rules configured by the different role classes. User requests should be as abstract as possible to leave enough degrees of freedom to the policy engine so that the probability of a successful mapping is maximized.

The following section introduces the concept of policy-enabled applications. It describes a software architecture for integration of applications and policies, shows the separation of policies into different role classes, and gives an insight how to deal with policy conflicts. It is also shown how to increase scalability by the use of abstract user requests. We present a specific distributed application scenario that deploys our concepts. Related work is described in Section 3, after which we draw our conclusions and motivate future work in Section 4.

2 Policy-Enabled Applications

Policy-enabled applications are a problem-domain–independent solution for administratively scalable distributed systems. This category of applications allows external entities to influence the behavior of the system by the definition of policies. Policy-enabled applications comply to these local domain policies. This section proposes the software architecture of policy-enabled applications, explores the different role classes of people that define policies, and finally explains how user requests processed by policy-enabled applications have to be structured to be most effective. Finally, we present an application scenario: a policy-enabled information system.

2.1 Policy-Enabled Software Architecture

Policy-enabled applications differ from normal applications in that their internal structure separates mechanism and protocol issues strictly from the policy part. Exploiting this concept enables them to process and comply to policy rules defined by entities outside the application.

The policy part of a policy-enabled application consists of a generic policy engine and a set of policy rules. The engine is able to evaluate these rules and to answer requests for policy decisions from the mechanism and protocol part of the application. For making these decisions the engine needs the support of external services: naming and directory services (e.g., DNS [Moc87], X.500 [CCI93]), location services (e.g., GlobeLS [SHB+98]), trust services (e.g., PolicyMaker [BFL96], SDSI/SPKI [RL96]), and services that control the policy engine itself and allow the maintenance of the policy rules. The decisions of the policy engine are also influenced by the environment of the application—the current time of day or the resources available at the target host. Finally, the policy engine must be able to enforce the taken policy decisions by the use of a policy-enforcement subsystem, in case it cannot trust in the decision conformance of the application behavior. The software architecture is displayed in Fig.2.1.

Every time the application code needs a policy decision (e.g., because it has received a user request), it constructs a policy decision request that is sent to the policy engine (e.g., "Is a certain user allowed to access?"). The policy engine consults its set of policy rules and makes a decision with the potential aid of external services and the consideration of environmental influences. The decision of the policy engine is then returned to the application code. Answers are normally simple boolean *true* or *false* statements, but can as well consist of labels (e.g., decision request: "Which encryption algorithm should I use?", decision: "3DES") or label sets (e.g., decision request: "Which encryption algorithms may I possibly use?", decision: "3DES or IDEA").

The policy rules range from simple expressions to modestly complex program statements written in a description language that is interpreted by the policy engine. The majority of the policy rules can be expressed as *if-then-else* cascades.

The communication between the policy engine and external services or the policy-enforcement subsystem can be done via standard APIs [MES99]. A generic interface between the application code and the policy engine is more

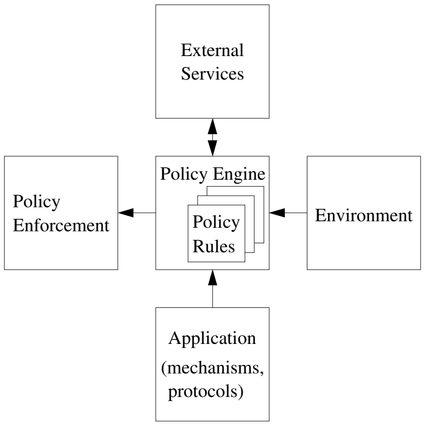

Fig.2.1 Policy-enabled application software architecture

difficult to define, because the application code has to include all available information into the policy decision request. The structure of the information in turn depends on the type of the request. The exact definition of the communication interfaces and the policy description language is beyond the scope of this paper and partially subject to further research.

2.2 Role Classes for Policies

Increasing administrative scalability of distributed applications means to strengthen the influence of various entities upon the behavior of the application. The fact that multiple influencing entities exist requires that each entity must be able to specify policies that express requirements according to its local needs. Thus the policy engine has to support the usage of a multitude of distinct policy rule sets.

To gain further insight into the characteristics of policy definitions, we have identified four different role classes that entities formulating policies play. Whereas traditional role concepts tend to classify entities in regard of their business duties, our definition is knowledge-based. Table 2.1 shows the responsibilities that define the role class, the knowledge that an entity playing a role has about the system, and the influence that it should be able to express in a policy definition.

Role Class Name	Responsibility	Knowledge	Policy Influence
Developer	designs and implements the components of the distributed application	mechanisms and protocols within the distributed system	maps the requirements that all other roles have onto concrete mechanisms and protocols
Application Administrator	installs and customizes the distributed application	structure and characteristics of the data that the distributed application uses	configures the distributed application so that it fulfills the global requirements uses mechanisms and protocols according to the data characteristics of the application

Table 2.1 Roles of entities defining policies

Role Class Name	Responsibility	Knowledge	Policy Influence
Domain Administrator	runs the components of the distributed application within the local domain	local platform (e.g., available hardware resources, network topology) local management demands usage patterns of local users	adapts the distributed application to comply to the local rules optimizes the behavior of the distributed application for local users
User	uses the application to get his/her work done	should not need any information about the structure of the application	should not need any influence on any policy

Table 2.1 Roles of entities defining policies

Note that the number of instances is different from role class to role class: there exists usually only one instance of the developer role class and also one of the application administrator role class. On the opposite, a large number of entities exist who play the domain administrator role class, and even more individuals playing the user role class.

From the different role classes different requirements for the architecture of a policy subsystem can be deduced. From the developer role class we learn that mechanisms must exist which control the modification of policy rules in a strict and secure way. As the requirements towards the administration of the application change frequently over the time of usage, mechanisms for the effective distribution of policy changes have to exist. Fast propagation of updated policy rules must be possible. We see that the aforementioned three dimensions of scale are not in fact orthogonal, because here again we have to deal with large numbers of policy subsystems or long distances between policy subsystems and therefore the numerical and geographical dimensions.

As noted earlier, a multitude of domain administrator role classes exist in large-scale distributed systems. This requires mechanisms for the support of tree-like hierarchical domain structures to allow decentralized policy management. Flexible administration models must be supported. These models range between two extremes: on one side we have a very strict model where the majority of the policy rules are located at the upper layers of the hierarchy or even at the root of the administrative tree. On the other side, loose administration models have to be supported where policy rules are set up in the leaf nodes with a very lean or even empty root and upper layer structure.

From the insight that users should not need any information about the internals of the application and should not need to define policies, requirements for the structure of user requests towards the distributed application can be deduced. These issues are discussed in further detail in Section 2.3.

Dealing with a large number of policy rules means also dealing with rule conflicts and situations in which no policy rule applies at all [Kue98]. Conflicts may arise between different role classes (e.g., between application administrator and domain administrator) or between entities playing the same role class (e.g., between root domain administrator and leaf node domain administrator). Mechanisms for the detection and solution of policy conflicts are the application of priorities and the employment of default policies in case of underspecification. These mechanisms must also be able to prevent a domain administrator from overriding the intentions of an application administrator. An application administrator must also not be allowed to break the semantics of the application by redefining policy rules set by the developer. Beyond the described static solutions, we consider the automatic solution of policy conflicts without any human interaction a very difficult yet interesting problem to solve.

2.3 Abstract User Requests

Extending the basic client-server paradigm towards distributed multi-server applications, we model the interaction between users and a distributed application as an interaction between clients and distributed servers. Clients and distributed servers—or users and a distributed application—communicate via user requests initiated by the clients. One example of such a user request is a WWW browser requesting a document with a certain URL from a WWW proxy network. Unlike traditional applications, the processing of a user request by a policy-enabled application is not directly accomplished by mechanisms and protocols. The request is rather mapped onto a proper set of mechanisms and protocols, where the mapping is accomplished by a policy engine. The policy engine makes its decision with the help of its configured of policy rules.

Systems scale poorly, when the percentage of unsuccessfully answered user requests rises with the growth of the system. Thus, one aspect of scalability is that the probability of a user request to be successfully processed is aimed to be very high. To process a user request successfully by a policy-enabled application, the policy engine needs to find a mapping of the request onto the available mechanisms and protocols. As the set of available mechanisms and protocols within each distributed application is limited, the probability of a successful mapping rises with the degree of freedom given to the policy engine. This degree of freedom depends again on two facts: the size of the decision space configured in the policy rules and the structure of the user request. The policy rules are configured by the role classes explained in Section 2.2 and are therefore in the majority of cases outside the scope of the application developer. So the user request has to give the policy subsystem as much degree of freedom as possible to allow the application to scale. A request giving freedom means in turn, that the request has to be as abstract as possible. Abstract requests do not specify exact information about the mechanisms and protocols with which they should be fulfilled. Instead, they only state what problem they want to be solved and not how. The whole correlation of requests, policies and mechanisms in terms of abstraction and degree of freedom is shown in Fig.2.2.

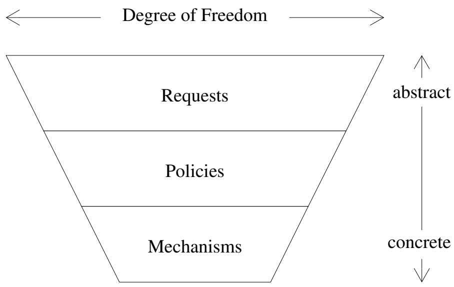

Fig.2.2 Grade of abstraction and degree of freedom of requests, policies and mechanisms

To give an example, from our model's view WWW URLs (Uniform Resource Locators) are badly specified requests. URLs contain both the exact location of the document (hostname and path information) and the protocol with which the document has to be retrieved. Thus, URLs place severe restrictions upon the distributed application, where the distributed application is the mesh of proxies, caches and web servers of which the WWW currently consists. The WWW has a very limited set of possibilities to fulfill a request for a specific URL because it is tightly bound to the specified mechanisms, protocols and locations. The current way of making the WWW scale better is to "cheat", because a cache server loads the requested document from its local disk instead of fetching it via HTTP as specified in the URL. We consider a location- and protocol-independent URN (Uniform Resource Name) scheme as proposed in [Sol98] to be more abstract and therefore more appropriate.

2.4 Scenario: Distributed Information System

We will show our concepts with an example of a distributed information system. The information system presents dynamic information data to its users—similar to the WWW—, is a multi-server system, and consists of several types of servers. Master servers hold primary versions of the information data. Information data can be replicated with the use of slave servers. These servers hold either complete copies of the information data or cache the results of previous queries answered by master servers. To accomplish this task, slave servers communicate with master servers via update protocols. A large number of update protocols have been proposed so far, so we will limit ourselves for simplicity reasons to the classification into server-push and client-pull update protocols. The third type of server is the proxy server or stub. A proxy server does not store any information data, it simply forwards queries to master or slave servers. It does not need to be notified in case of data modifications and therefore does not participate in any update protocols.

To access information data contained in the system, a client has to instantiate a stub or proxy server on its local machine similar to CORBA [OMG98], Java RMI [Sun97], and AspectIX [GSB+98]. Clients should not need to know the locations of master and slave servers. The topology should be transparently hidden from the clients. Topology modifications (e.g., to adapt the system to changed access patterns) should be possible without notifying any clients.

A developer can now design and implement these components, but unless he is able to predict the structure of the information data, the network topology, and user requirements, he is unable to make proper decisions about the usage of those components. When applying the concepts of policy-enabled applications, he delegates a decision like "To which server should my proxy connect to?" to a policy engine instead of wiring the decision making into the application code. Then the policy engine is able to make that decision based on the current set of of policy rules. The developer is not only responsible for the separation of mechanism and policy code, he also defines the basic policy rules which describe the facilities of the components (e.g., available update protocols, maximum number of slave servers). In this step, the developer should describe only what the components are able to do, not what the developer thinks they should be doing, as this would unneccessarily limit future use under changed conditions.

As described in Section 2.2 other external entities are now able to define their own policy rules. An application administrator who knows the overall system requirements (e.g., structure and average size of information data, modification rate of information data) and who is responsible for the functioning of the application, expresses an own set of policies that is also loaded into the policy engine. Various domain administrators with their specific requirements (e.g., the existence of a high speed network within the domain with a slow link to the outside world) define local domain policy rules that are also taken into account when a decision has to be made.

A possible deployment scenario of the distributed information system is shown in Fig.2.3.

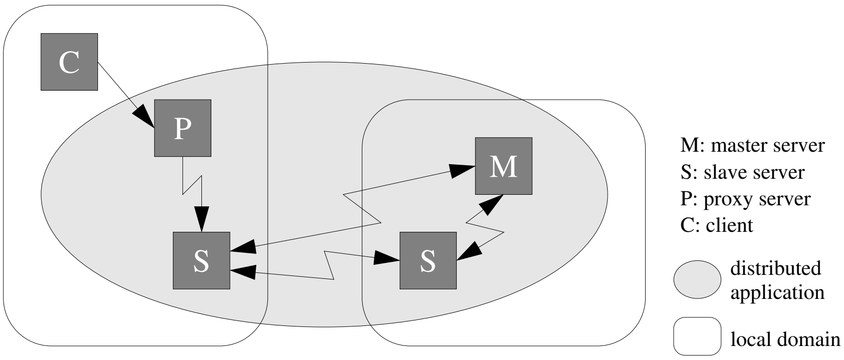

Fig.2.3 Distributed information system

Table 2.2 shows example policy rules defined by the different role classes.

The first two policy rules defined by the domain administrator clearly show that a classification into mandatory ("must") and discretionary ("if possible") policy rules is needed. If the second rule would read "proxy must connect to master or slave server in my domain" instead, the degree of freedom of the overall rules would be severely limited.

Role Class Name	Policy Rules and User Requests
Developer	only one master server per system number of slave servers must not exeed Max_S if Max_{Age} < 30 minutes then use server-push update protocol else use client-pull update protocol
Application Administrator	if at least Min_{Res} resources available then deploy slave server information data must not be older than Max_{Age}=60 minutes
Domain Administrator	if possible deploy master or slave server in my domain if possible proxy connects to master or slave server in my domain slave server must not use more than Max_{Res} resources
User	information data must not be older than Max_{Age}=10 minutes

Table 2.2 Policies and user requests defined by different role classes

Requests for specific information data are what we named user requests. But note that the rule set by the user is not a policy rule but a special kind of user request. The user does not demand neither an explicit update protocol to be used nor a specific server topology. His only interest is that the retrieved information data has a certain actuality. This makes the user request highly abstract and gives the policy engine many degrees of freedom for the actual decision to make. The policy engine is now able to decide how to answer this user request successfully—possibly through replacing update protocols or reconfiguring the server topology.

Other deployment examples of policy-enabled applications are: security properties (e.g., access control, trust issues) of distributed systems controlled by policies as demanded in [LSM+98], or policy-based resource usage control (e.g., quality of service parameters).

3 Related Work

The majority of recent policy research was made in the field of distributed systems management. The main focus lies in the integration of policy capable mechanisms directly into the management systems [Slom94] [SlTw94] [MAP96] [MaCa93]. They deal with problems like the representation of policies with the help of dedicated languages [KKK96] [Moff94], policy hierarchies [MoSl93] and their mapping into concrete management actions [Wies95], and the treatment of policy conflicts [LuSl99]. For accessing policy capable management systems, for the classification of management actions into categories and for the concrete execution of management decisions, role-based mechanisms are used [FeKu92] [LuSl97]. Our approach differs from all these projects in the fact that they concentrate on management issues. Consequences of the employment of policies for the structure of the managed applications are not covered. From their point of view, all activities are triggered by the management system so the managed application sees these activities as asynchronous events. In our model, the managed application itself invokes the policy engine synchronously whenever a policy decision is needed.

An IETF working group currently designs a framework for policy-controlled networks [MES99]. Their approach focusses on the definition of protocols for the interaction between policy engines and policy-enforcement subsystems (policy decision points and policy enforcement points in current IETF terminology [SE99]) and the definition of storage schemes for policy rules in LDAP-capable databases [MES99a]. The design of a policy definition language PDL [SS98] is also underway, but the expressiveness of the currently drafted PDL concentrates on the management of network devices and quality of service in networks. Multiple policy domains are not supported, entities defining policies cannot be assigned to specific roles and because of the very focussed view of the work there is limited abstraction available for the definition of policy rules. We consider using the protocols and schemes defined by the IETF within our policy-enabled application software architecture when their development has finished.

An interesting work about the relationship of policies and trust can be found in [Ess97]. Their definition of policies is influenced from a more business-oriented view that leads to a relatively abstract policy description. Role class policy definitions and the mapping of abstract policies onto concrete mechanisms is yet not an issue.

Other work on policy systems can be found in [Bro98], [ABG+98], and [Gut97], but their focus is always limited to a specific problem domain (e.g., security policies, routing policies, filtering policies) so they do not describe a generic approach as we do.

4 Conclusions and Future Work

We have presented the concept of policy-enabled applications, which increases the administrative scalability of distributed applications. Policy-enabled applications allow external entities to control the application behavior by defining policy rules to express their intentions. We have identified four different role classes of entities that define policies: developer, application administrator, domain administrator and user. The definition of default policies and the application of priority schemes help solving policy conflicts.

In our model, user requests that have to be processed by a policy-enabled application are mapped onto concrete mechanisms and protocols by a policy engine. To maximize the probability of a successful mapping, we demanded that the user request must leave many degrees of freedom to the policy subsystem, and thus has to be as abstract as possible.

The more influence external entities can exert upon a distributed application, the higher acceptance this application gains among administrators and users. Together with a high probability of successful user request processing through the use of abstract user requests, policy-enabled applications exhibit a superior administrative scalability compared to traditionally designed ones.

One of our future issues will be the exact definition of the communication interfaces between the components of policy-enabled applications. We will also finish the definition of our policy description language. Other problems to be tackled in the future include multi-domain issues (e.g., the usage of local names in policy rules, compatibility of names in policy rules) and the analysis of software structures where mechanism and policy code is not trivially separable.

References

ABG+98 C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra: *Routing Policy Specification Language (RPSL)*, RFC 2280, January 1998

Bro98 G. Brose: “Towards an Access Control Policy Specification Language for CORBA”, In *Proc. of the ECOOP 98 Workshop on Distributed Object Security*, Brussels, 1998

BFL96 M. Blaze, J. Feigenbaum, J. Lacy: “Decentralized Trust Management”, In *Proc. of the 1996 IEEE Symp. on Security and Privacy*, pp. 164-173, IEEE Computer Security Press, Los Alamitos, Cal., May 1996

CCI93 CCITT: *The Directory—Overview of Concepts, Models and Services*, CCITT X.500 Series Recommendations, 1993

Ess97 D.J. Essin: “Patterns of Trust and Policy”, In *Proc. of the New Security Paradigms Workshop ’97*, pp. 38-47, Langdale, UK, September 1997

FeKu92 D. Ferraiolo, R. Kuhn, R: “Role-based access control.” In *Proc. of the 15th National Comp. Security Conf.*, 1992.

Gut97 J.D. Guttman: “Filtering Postures: Local Enforcement for Global Policies”, In *Proc. of the 1997 IEEE Symp. on Security and Privacy*, pp. 120-129, IEEE Computer Society Press, Los Alamitos, CA, May 1997

GSB+98 M. Geier, M. Steckermeier, U. Becker, F. Hauck, E. Meier, U. Rastofer: “Support for mobility and replication in the AspectIX architecture”, In *Object-Oriented Technology, ECOOP’98 Workshop Reader*, pp. 325-326, LNCS 1543, Springer, September 1998

KKK96 T. Koch, C. Krell, B. Krämer: “Policy definition language for automated management of distributed systems.” In *Proc. of 2nd Int. Workshop on Sys. Management*, IEEE Comp. Society, Toronto, June 1996.

Kue98 W.E. Kühnhauser: “A Classification of Interdomain Actions”, In *Operating System Reviews* 32(4): 47-61, ACM, October 1998

LSM+98 P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, J.F. Farrell: “The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments”, In *Proc. of the 21st National Information Sys. Security Conf.*, October 1998

LuSl97 E. C. Lupu, M. S. Sloman: “Towards a role based framework for distributed systems management.” In *J. of Netw. and Sys. Management 5(1)*, Plenum Press, 1997.

LuSl99 E. C. Lupu, M. S. Sloman: “Conflicts in policy-based distributed systems management”, In *IEEE Trans. on Softw. Eng. — Special Issue on Inconsistency Management*, 1999.

Neu94 B.C. Neumann: “Scale in Distributed Systems”, In *Readings in Distributed Computing Systems*, IEEE Computer Society Press, 1994

MaCa93 M. J. Masullo, S. B. Calo: “Policy management: an architecture and approach.” In *Proc. of IEEE Workshop on Sys. Management*, UCLA, Cal., April 1993.

MAP96 B. Meyer, F. Anstötz, C. Popien: “Towards implementing policy-based systems management.” In *IEE/IOP/BCS Distr. Sys. Engineering J. 3(2)*, 1996, pp. 78–85.

MES99 B. Moore, E. Ellesson, J. Strassner: *Policy framework core information model*. Internet-Draft, Work in Progress, June 1999.

MES99a B. Moore, E. Ellesson, J. Strassner: *Policy framework ldap core schema*. Internet-Draft, Work in Progress, June 1999.

Moc87 P.V. Mockapetris: *Domain Names—Concepts and Facilities*, RFC 1034, November 1987

Moff94 J. D. Moffet: “Specification of management policies and discretionary access control.” In M. S. Sloman (ed.): *Management of Distr. Sys. and Comp. Netw.*, Addison-Wesley, 1994, pp. 455–480.

MoSl93 J. D. Moffet, M. S. Sloman: "Policy hierarchies for distributed systems management." In *IEEE J. of Sel. Areas in Comm. 11(9)*, 1993, pp. 1404–1414.

OMG98 Object Management Group: *The Common Object Request Broker Architecture and Specification*, Revision 2.2, OMG Document formal/98-02-01, OMG, Framingham, Mass., 1998

RL96 R.L. Rivest, B.W. Lampson: "SDSI—A Simple Distributed Security Infrastructure", MIT, Mass., 1996

SHB+98 M. van Steen, F.J. Hauck, G. Ballintijn, A.S. Tanenbaum: "Algorithmic Design of the Globe Wide-Area Location Service", In *The Computer Journal* 41(5):297-310, 1998

SlTw94 M. S. Sloman, K. P. Twidle: "Domains: a framework for structuring management policy." In M. S. Sloman (ed.): *Management of Distr. Sys. and Comp. Netw.*, Addison-Wesley, 1994, pp. 443–453.

Sol98 K. Sollins: *Architectural Principles of Uniform Resource Name Resolution*, RFC 2276, January 1998

Slom94 M. S. Sloman: "Policy driven management for distributed systems." In *J. of Netw. and Sys. Management 2(4)*, Plenum Press, 1994.

SS98 J. Strassner, S. Schleimer: *Policy Framework Definition Language*, Internet Draft, Work in Progress, November 1998

SE99 J. Strassner, E. Elleson: *Terminology for describing network policy and services*, Internet Draft, Work in Progress, February 1999

Sun97 Sun Microsystems: *Java Remote Method Invocation Specification*, Whitepaper, Mountain View, CA, 1997

Wies95 R. Wies: "Using a classification of management policies for policy specification and policy transformation." In *Proc. of the IFIP/IEEE Symp. on Integrated Netw. Management*, Santa Barbara, Cal., May 1995.