A Comparative Study of Policy Specification Languages for Secure Distributed Applications

Sandrine Duflos[1], Gladys Diaz[2], Valérie Gay[1], and Eric Horlait[1]

[1]LIP6, Université P. & M. Curie, 8 rue du Capitaine Scott, 75015 Paris, France,
{Sandrine.Duflos, Valerie.Gay, Eric.Horlait}@lip6.fr
[2] L2TI, Université Paris 13, 99 Av. J.-B. Clément, 93430 Villetaneuse, France,
Gladys.Diaz@galilee.univ-paris13.fr

Abstract. This paper presents a comparative study of policy specification languages. Our objective is to find policy language or notation that is the most suitable to express the security aspects of distributed applications running on policy-based networks. We first made a selection of languages and we compare them on several criteria: their suitability to specify security, their ability to express both user and network oriented security aspects, the representation technique they use and the notions they are able to express. This paper concludes on a discussion on what would be the ideal policy language for distributed applications that have strong security constraints.

1 Introduction

The exponential use of distributed applications such as electronic commerce, video on demand and videoconferencing increases the complexity of control and management of resources provided by their networked environment.

On one hand, the users and developers of those applications are aware of the requirements of their applications, in particular in terms of security and are willing to express them for the environment to take them into account. On the other hand the networks, in particular policy-based networks that are at the centre of our study, are more and more able to deal with all type of traffic and to provide a service 'a la carte'. The bottleneck in this situation is a comprehension problem between the users and the network.

Our objective is to find a way to specify security that is suitable to the users, the applications and their policy-based networked environment. To achieve this goal, we have selected the following security-oriented specification languages: ASL (Authorisation Specification Language) [18], Automated manager [31], DLSS (Deontic Logic-based Security Specification) [6], ISPS (IPsec Security Policy Specification) [12], LaSCO (Language for Security Constraints on Objects) [17], Ponder [7], SPSL (Security Policy Specification Language) [4] and Tower [16].

In addition to those security-oriented languages, we chose to study some QoS oriented languages. Indeed, security aspects of distributed applications are very often included

M. Feridun et al. (Eds.): DSOM 2002, LNCS 2506, pp. 157–168, 2002.

in the Quality of Service (QoS) parameters. Therefore, to this extend, QoS-oriented languages allow to specify security requirements. The selected QoS-oriented specification languages are: HQML (Hierarchical QoS Mark-up Language) [15], PDL (Policy Description Language) [2], [22], [34], PDN (Policy Definition Notation) [26], [27], PPL (Path-based Policy Language) [33], QML (Quality of service Modelling Language) [9], [10], [11], [21], QuAL (Quality of service Assurance Language) [14], [36] and QuO (Quality Objects) - QDL (Quality Description Language) [23].
Some of those languages are not 'policy' languages (HQML, QML, QuAL, QuO-QDL) but they have been selected because they contain features that we would like to find in an ideal policy language whose aim is to support application with strong security constraints.

This paper compares those languages on four important criteria. They are:

- the *suitability to specify security* (Section 2). This criterion is essential since security is at the centre of our concern,
- the *existence of different abstraction levels* to represent policy and the *possibility to map* between these different levels (Section 3). This is an important feature for the language we need. It enables to bridge the gap between the user and its supporting environment,
- *the representation technique used* by the language to represent the policies (Section 4). This criterion will have an influence on the policy expressiveness for the different users (end-users, administrators, etc.),
- the *notions used* for the organisation of information in the language (Section 5). This gives an indication of the complexity of the language.

Section 6 concludes on the interesting features that would be good to put together to create a complete policy language for secure distributed applications.

2 Comparison on the Suitability to Specify Security

This section studies the languages on their ability to specify the following security aspects:

- *Can access control be expressed?* In this case, rules are defined to specify under what conditions a user or an element can access resources, information or other elements of the system. The majority of the selected security-oriented languages propose the specification of access control policies. These policies are often expressed in terms of positive and negative authorisations as in Automated Manager, Ponder, ASL and DLSS. DSSL differentiates the authorisation and the permission to execute an action. An authorisation may be necessary, but not sufficient to get an explicit permission. In other words a user can be authorised to read pdf files but the owner of the file does not permit the access. Tower is based on

the RBAC (Role-Based Access Control) approach. In that approach security constraints are related to the roles played by the users and policies are expressed through permissions and privileges. The permissions are the conditions on which the security is applied and the privileges are the actions to execute. Other languages such as SPSL and ISPS provide another access control viewpoint with a fire walling role that stop non-authorised traffic.

- *Can identification/authentication be provided?* Users are identified with their group or associated role – these notions are explained in more details in Section 5 –, in other words at the entity creation within the system. They can also be identified within policies. For example in Ponder, it is possible to verify if the user belongs to a group within an authorisation policy: if *belongs (user, group)* then *action is authorised*. However none of these languages provide the authentication. Only SPSL permits to create policies with this aspect but for communication security. It also provides another kind of authentication: the one of the persons who maintain the policies.
- *Can confidentiality and integrity be expressed?* Those aspects are bound to communications and are available in SPSL and ISPS languages. They permit to express sets of security parameters that are required to secure the communication. They can be the algorithms used for encryption, the strength of encryption, the source and destination of the message, etc.
- *Can obligation and prohibition be expressed*? These notions state that a user must or must not execute or be allowed to execute an action on an element as in ASL, Automated Manager, DLSS, PDL and Ponder. They force or forbid users to execute actions on elements. For instance, if *user1 is not a doctor* then *read (medical records) is prohibited.*
- *Is there a system audit?* Obligation and refrain policies introduced in Ponder can react to system events such as time events or to events that are composed of specific operators. An example: if *student connection after 8:00 p.m.* then *refrain connection*. Here the event is the connection of the student after 8:00 p.m. This kind of policy can also be used in QoS management: if *performance degradation* then *reserve bandwidth for user members of PremiumService*. Automated Manager, DLSS, LaSCO and PDL policies can also be enforced using the system audit.
- *Is there a delegation technique?* Delegation temporary transfers privileges to other user. This type of policy has a particular form, which can be: delegate *set of privileges* from *user1* to *user2* during *time t*. Only Ponder and DLSS provide this feature.

Table 1 summarises the results.

Ponder is the most complete language to express security policy. An example of security specification in this language is given in Table 2. It shows how to express the fact that there is a need to strongly secure an online payment through the use of an obliga tion policy (*oblig* keyword). The obligation policy reacts to an event (following

Table 1. Comparison on the suitability for security specification

Languages	ASL	Automated Manager	DLSS	Goal-Based	ISPS	LaSCO	Ponder	PPL	SPSL	Tower
access control	✓	✓	✓	✓	✓	✓	✓	✓	✓	RBAC
identication - authentication	✓	✓				✓	✓		✓	✓
integrity - confidentiality					✓				✓	
audit			✓			✓	✓			
delegation			✓				✓			
obligation - prohibition	✓	✓	✓				✓			

the *on* keyword) that enables to detect the execution of an online payment from the client identified by its *clientid*, to the merchant identified by its *merchantid*. This policy enforcement is required by the subject that must be an end-user and is applied to the target (the traffic). The action to do follows the *do* keyword. Two actions are specified and separated by '|'. This operator indicates that the second action (error()) is performed only if the first action fails. The purpose of the first action is to strongly secure the communication between the client and the merchant.

Table 2. Ponder syntax example

```
inst oblig SecureOnlinePayment {
  on          OnlinePaymentUse(clientid, merchantid)
  subject     s = /EndUser
  target      t = /Traffic
  do          securecommunication(clientid, merchantid, strong) | error()
}
```

3 Comparison on the Level of Abstraction and the Existence of Mapping and Feedback Mechanisms

Security parameters and security policies can be specified at different level of abstraction.

At the user and application levels (high level), the security specification enables to control the execution of applications and to specify the conditions and rights for the interaction between the users and their applications. High level policy generally refers to a declarative statement of what the end-user wants (a proposition or a single action). An example of high level policy is given in Table 3.

Table 3. High level policy example

I (Sandrine) require a strong security protection for my e-commerce online payment.

At the middleware and network levels (low level), the security specification enables to represent and to verify the constraints in the supporting environment. Low level policy representations are generally procedural and specify the logic of how to achieve the goal (as rules or program control flows that are evaluated to determine a sequence of actions that should be taken). An example is given in Table 4. The syntax of this policy is detailed in [13]. This low level policy is the result of a mapping of the high level policy of Table 3 in a particular environment. This environment supports the IPsec architecture and secures distributed application exchanges with policies. The policy depicted here requires the use of the IPsec ESP protocol and proposes several algorithms (3DES for confidentiality and SHA-1 or MD5 for integrity), the tunnel mode and the anti-replay protection, to secure the communication between the IP addresses 2.2.2.2 and 3.3.3.3. The application is identified by the port number (8000) the communication direction (bi-directional) and the configuration type used (unicast).

Table 4. Low level policy example

IF SourceIPaddress = *2.2.2.2* and SourcePortNo = *8000* and DestinationIPAddress = *3.3.3.3* and DestinationPortNo = *8000* THEN CONNECT with *bi-directional* and *unicast* among *2.2.2.2* at *8000* and *3.3.3.3* at *8000* with *AF11* and *ESP* with *(3DES)* and *(SHA-1, MD5)* and *tunnel* and *Anti-Replay*

Abstraction levels differs from a language to another.

All the languages we studied in this paper are able to specify security at middleware/network level. However, to be really suitable for distributed applications security specification, they also need to be able to express user and application security requirements. This supposes the existence of a user/application level of abstraction. HQML, QML, QuAL and QuO-QDL have that characteristic.

In addition to the different level of abstraction, there is a need for a good mapping mechanism to go from one level to another. Policy classifications [28], [35], [38] and policy hierarchies [39] are ways to do this. Amongst the languages we have studied, only HQML, ISPS, QML, QuAL have those mechanisms. For example, in QML the refinement process is done using hierarchy between interfaces definitions (contract and profile type are refined in contracts and profiles instances). Automated Manager proposes a manual mapping mechanism.

Ideally, the mapping mechanisms should be both ways since a feedback is necessary for the application to be able to adapt changes in its environment. If we go back to the example in Table 4, a feedback from the network could be: the 3DES algorithm is not available for the ESP protocol; and it could lead to the cancellation of the transaction specified in Table 3 since securing the communication becomes impossible. From the languages we studied, only HQLM proposes that feature.

Table 5. Comparison on the different level of abstraction and mapping and feedback possibilities

Languages	ASL	Automated Manager	DLSS	HQML	ISPS	LaSCO	PDL	PDN	Ponder	PPL	QML	QuAL	QuO-QDL	SPSL	Tower
User/application level				✓							✓	✓	✓		
Middleware/network level	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mapping mechanisms		manual		✓	✓						✓	✓			
Feedback mechanisms				✓											

4 Comparison on the Policy Representation Techniques

In this section we present how the policies are represented in the different languages. This is important to identify the expressiveness and the flexibility of each one of them. Different approaches exist to specify policies. As mentioned in [7] we can group all the languages into three major approaches: a specification through a *policy language*, a specification through *policy rules*, and a specification through a *logical language*. The use of a language that is specifically made to express policy provides a considerable flexibility compared to the other approaches. The rule-based specification is generally linked to the IETF policy models [32], [29] and expresses policies through the following form: *if E then A*, where *E* represents some detectable event or network state, and *A* represents an action to be taken when E is detected. This kind of specification often specifies low level policies. Finally the logical languages (deontic, real-time or modal logic models) use the logical operators to represent the policies and is more complex to understand.

In addition to the approach, there are different techniques to represent the policies:

- *Functional blocks:* they are used to describe the class of policies or the application and communication constraints. These classes are represented through specific structures like in PDN, QuAL, QuO and Tower, through object classes like in Ponder, QML and SPSL or through the use of a markup language syntax like in HQML.
- *Notation:* these languages have developed a specific notation that is not linked to a specific structure (object, functional block, etc.).
- *Graphs:* in this case policies are represented using direct graphs with annotations. The graph representation is used in several languages. LaSCO is completely based on graphs: the system to which the policy is applied consists of a series of events, each occurring between a pair of objects. A policy is applied through the identification of portions of a system to which policy applies. The requirements are

checked on that portion of the system. Other models use graphs to represent a part of their policy specification. For example in HQML, graphs are used to represent the application configurations.

- *Path-based:* the path based category indicates that is possible to specify in the policy the element composing the route as in PPL, or that can act on the traffic flow as in ISPS.

In addition some languages are based on existing standard modelling language such as QML that is based on the UML (Unified Modelling Language) model or HQML that is specified in XML (Extensible Markup Language).

Table 6 summarises this comparison.

Table 6. Comparison on the policy representation techniques

Languages		ASL	Automated Manager	DLSS	ISPS	LaSCO	PDL	PDN	Ponder	PPL	SPSL	Tower
Approach	Policy rules				✓		✓			✓	✓	
	Policy language		✓			✓			✓			✓
	Logic-based	✓		✓								
Technique	Functional blocks							✓	✓		✓	✓
	Notation		✓	✓	✓		✓			✓		
	Graphs					✓						
	Path-based				✓					✓		

5 Comparison on the Notions Used

In this section, we study the different notions used in the languages. Synonyms and homonyms have been found in the different specification languages.

A first notion we found is the group. Three types of groups can be distinguished:

- *Domain:* A domain generally groups elements together (objects in Automated Manager, PDN and Ponder and network entities in PPL and SPSL) in order to apply a common set of policies on them. One main goal is to reduce the number of policies for scalability problem. A set of policies is not associated any more to an element but a group of elements. Elements can be grouped together according to geographical boundaries, object type, responsibility and authority or at the convenience of the managers. LaSCO introduces another definition of the domain notion. This language describes a domain as a specific system situation (e.g. a

specific user doing a specific action on a specific target) under which the policy is applied. A domain is associated to a single policy. A similar definition is given in the Goal-Based model with the Goal Domain notion [1].

- *Group of User:* This notion is found in ASL. It is closely related to the notion of domain. It groups users together and the same policies are applied to the whole group. The major difference between the notion of group of users and domain is that the policies are included into the group. That is not the case for the domain.
- *Role*: A role is linked to a set of rules describing the rights and duties associated to it. It refers to a position in an organisation. The position is often a human job function. We find this feature in ASL and Tower. In these two languages, a role is related to a particular user and can be activated and deactivated. However several users can play the same role and a user can have several roles. In ASL roles are organised hierarchically. Others languages permit to refer, in addition to job functions, to positions as software or hardware (e.g.: core-router) like in Automated Manager or Ponder. In these languages the role is associated to a domain. In DLSS, the object class notion is closely related to the role. The object class represents a group of objets and contains a set of methods that describes the actions that are authorised. The Ponder language describes another mean to group policies together: the group. A group represents a set of policies and constraints with some semantic relationship (e.g. relating to a particular application). More complex notions are also introduced such as the relationship – to create relations between roles – and the management structure – to represent for example a university department by grouping together all the groups, roles and relationships concerning this department –.

In addition to the group, the following notions have been identified:

- *Goal.* A goal enables to define the policies in relation to the objectives to be met. A desired behaviour or a state can represent these objectives in the system. In ISPS the word 'goal' is replaced by 'requirement' and refers to a high level objective. In PDL a goal level is defined to represent policies. Goal oriented level enables specifying objectives to meet in using templates of attributes specifying events, constraints and actions. A specific model described in [1] is completely based on this notion. In the Goal-Based model, a policy is defined as a collection of policy goals and/or policy rules that express certain desired system behaviour. Here there is a difference between goals and rules. A policy goal specifies what system behaviour is desired, that is, goals describe the desired system state, but not how to reach the desired state. A policy rule specifies action(s) to be taken to reach a desired state.
- *Profile*. A profile represents the particular values (preferences or configurations) of some parameters. These configurations can be applied to users, applications or resources. In Policy-Driven [25] this notion is related to the user expectations. In QML, QoS profiles enable to correlate the application configurations that match the user's requests and that could be supported by the client's current resource conditions. QML profiles associate contracts with interfaces, operations, opera-

tion arguments and operation results. In QuO framework the notion of profile is used to define the contract between the client and an object. In HQML, profiles are realised through the application configuration.

- *Contract*. In general a contract specifies an agreement between clients and servers. Different definitions have been found: in QML a contract represents a particular QoS specification within a given category. In QuO a contract between the client and an object specifies the level of service desired by the client, the level of service the object expects to provide, operating regions indicating possible measures, and actions to take when the level of service changes.

Table 7 summarises the results.

Table 7. Comparison on the notions used in the languages

Languages		ASL	Automated Manager	DLSS	HQML	ISPS	LaSCO	PDL	PDN	Ponder	PPL	QML	QuAL	QuO - QDL	SPSL	Tower
Groups	Domains		✓				✓		✓	✓	✓				✓	
	of Users	✓														
	Roles	✓	✓	✓						✓						✓
Goals						✓		✓								
Profiles					✓							✓	✓	✓		
Contracts												✓	✓	✓		

6 Conclusion

For distributed application such as e-commerce the best language does not exist yet. It should be a mix of

- Ponder, ISPS and SPSL for the suitability for specification of security. Ponder is the more complete language. However it cannot express the communication security unless calling a predefined function. For the security of communication ISPS and SPSL are available. SPSL is the most suitable for IPsec policy specification. But ISPS is easier to understand.
- HQML for its different abstraction level to represent policy, the ability to map between these different levels and to give feedback to the higher level of abstraction. HQML is the only language from our selection that proposes an automatic mapping. Others such as PPL will consider the mapping in their future work. Languages such as PDL and QuO, originally system- or network-oriented, begin to implement mechanisms to translate low level information into application level information and therefore give feedback to the application and users. All the languages we selected use notations and parameters requiring technical knowledge. They are therefore not really understandable to an end-user.

- Ponder, ISPS and PPL for the technique used to represent policies. Ponder is interesting due to its policy language approach and its specific policy objects that provide a flexibility, an abstraction and a completeness in the specification. However, the rule based approach is interesting at network level when the interaction with standard policy models is required. The 'mapping' is easier. Finally the most user-friendly is LaSCO with its graph-based policy representation but the system management stays quite complex.
- Of all the notions we studied, the most important is the notion of group (in particular for domain and role). Defining domains by grouping together the elements (objects, users, etc.) permits to reduce the number of policies. Roles enable the reuse of policies. These two notions, found in Automated Manager and Ponder, contribute to the scalability of large systems.

The need of policy detection conflict is also important. Among the studied languages, ASL, ISPS, and Ponder propose this functionality. ASL proposes specific conflict resolution policies. In ISPS, a conflict is resolved using algorithms that refine requirement policies into implemented policies. In Ponder, meta-policies are used to forbid the simultaneous execution of conflicting policies. Currently, there is a lack of automatic method for checking policy conflicts. Ways to detect statically potential conflicts are reported in [39], [24].

Another important criterion for the security specification is the policy release. This permits to identify how policies are triggered in the selected languages. The languages can be classified as *Proactive* or *Reactive*. In proactive languages the policies are represented by the condition(s) to verify to execute the action(s). Therefore, potential problems are detected and handled before they actually happen [25]. In reactive languages the policies are triggered according to the systems events occurrence. These are event-based languages. The policies can be triggered when a security violation occurs. The majority of languages we selected represent policies with a proactive approach. LaSCO, PDL and Policy Driven are examples of reactive languages. Currently, with the complexity of environment and applications that run it, the capability of languages to specify both proactive and reactive mechanism is essential. PDN, QuAL, Ponder and DLSS have the capability to trigger the policies using both proactive and reactive approaches. For our ideal language, we would choose such a language.

In short, we would choose Ponder as a basis since it is the most complete. It permits security, QoS and more general policy specification and we would add specific policies for the security of communication, the possibility to express high level policies and mapping and feedback mechanisms. It would not be the ideal language but would be a good alternative.

References

1. Bearden M. et al.: Integrated Goal Specification in Policy-Based Management. In: Proc. of Policy Workshop, Lecture Notes in Computer Science, Vol. 1995. Springer-Verlag (2001)

2. Bhatia R. et al.: Policy Evaluation for Network Management. In: Proc of INFOCOM 2000, Tel-Aviv, Israel (2000)
3. Casassa Mont M. et al.: POWER Prototype: Towards Integrated Policy-Based Management. In: HP Labs Technical Reports HPL-1999-126 (1999)
4. Condell M. et al.: Security Policy Specification Language. In: IETF Internet-draft (2000)
5. Corba Security Service Specification v 1.7. In: OMG ftp://ftp.omg.org/pub/docs/formal/01-03-08.pdf (2001)
6. Cuppens F. et al.: Specifying a Security Policy: A Case Study. In: Proc. of the computer security foundations workshop, Kenmare, Ireland, (1996)
7. Damianou N.: A Policy Framework for Management of Distributed Systems. In: Ph.D. Thesis. Department of Computing, Imperial College (2002)
8. Diaz. O.: QoS Policy Specification – A Mapping from Ponder to the IETF Policy Information Model. In: Proc. of ENC01, Aguascalientes, México (2001)
9. Frolund S. et al.: QML: a language for Quality of Service specification. In: HP Labs Technical Report, HPL-98-10 (1998)
10. Frolund S. et al.: Quality of Service Aware Distributed Object Systems. In: HP Labs Technical Report, HPL-98-142 (1998)
11. Frolund S. et al.: QRR (QML Runtime Representation). In: HP Labs Technical Report, HPL-98-159 (1998)
12. Fu Z. et al.: IPsec/VPN Security Policy: Correctness, Conflict Detection, and Resolution. In: Proc. of Policy 2001, Lecture Notes in Computer Science, Vol. 1995. Springer-Verlag (2001) 39-56
13. Gay V. et al., Policy-Based Quality of Service and Security Management for Multimedia Services on IP networks in the RTIPA project. In: Proc. of MMNS 2002, Lecture Notes in Computer Science, Springer-Verlag (2002)
14. Gomes P. et al.: Management of Application Quality of Service. In: Proc. of DSOM 94 Toulouse, France (1994)
15. Gu X. et al.: An XML-based Quality of Service Enabling Language for the Web. In: Research Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaing, UIUCDCS-R-2001-2212 (2001)
16. Hitchens M. et al.: Tower: A Language for Role Based Access Control. In: Proc. of Policy 2001, Lecture Notes in Computer Science, Vol. 1995. Springer-Verlag (2001) 88-106
17. Hoagland J. et al.: Security Policy Specification Using a Graphical Approach. In: Technical report CSE-98-3, University of California, Davis Department of Computer Science. (1998)
18. Jajodia S. et al.: A Logical Language for Expressing Authorizations. In: Proc. of the Symposium on Security and Privacy 1997, IEEE Press, (1997) 31-42
19. Koch T. et al.: Rules and agents for automated management of distributed systems. In: IEE/BCS/IOP Distributed Systems Engineering, Special Issue on Management, Vol 2, (1996)
20. Koch T. and Krell C.: Policy Definition Language for Automated Management of Distributed Systems. In: IEEE Computer Society (1996)
21. Koistinen J. et al.: QoS negotiation algorithm for QML. In: HP Labs Technical Report, HPL-98-51R1 (1998)
22. Lobo J. et al.: A Policy Description Language. In: Proc. of AAAI'99, Orlando, Florida (1999)
23. Loyall J.P. et al.: Specifying and Measuring Quality of Service in Distributed Objects. In: Proc. of ISORC '98, Kyoto, Japan (1998)
24. Lupu E.C. and Sloman M.: Conflict analysis for management policies. In Proc. of IM97, San Diego, CA, USA (1997) 430-443

25. Lutfiyya H. et al.: Managing QoS Requirements. In: UWO Technical Report 547 (1999)
26. Meyer B. et al.: Defining Policies for Performance Management in Open Distributed Systems. In: Proc of. DSOM'94 Toulouse, France (1994)
27. Meyer B. et al.: Flexible management of ANSAware applications. In: Proc. of ICODP95, Brisbane, Australia (1995) 255-265
28. Moffett J. D.et al.: Policy Conflict Analysis in Distributed System Management. In: Journal of Organizational Computing (1993)
29. Moore B. et al.: Policy Core Information Model -- Version 1 Specification. In: RFC 3060 (2001)
30. Object Management Group. Object Constraint Language Specification. In: chapter 7 in OMG Unified Modeling Language Version 1.3 (1999)
31. Sloman M. et al.: Policy Specification for Programmable Networks. In : Proc of IWAN'99, Lecture Notes in Computer Science, Springer-Verlag, Stefan Covaci (ed.) (1999)
32. Snir Y. et al.: Policy QoS Information Model. In: IETF internet draft (2001)
33. Stone G.N. et al.: Network Policy Languages: A Survey and a New Approach. In: IEEE Network (2001)
34. Virmani A. et al.: Network Management for the SARAS Softswitch. In: Proc. of NOMS2000, J. Hong, R.(ed.), Weihmayer, Hawaii (2000) 803-816
35. Vogel A. et al.: Distributed Multimedia and QoS - A Survey. In: IEEE Multimedia, Vol2, No2 (1995) 10-19
36. Wang. P.Y. et al.: Experimental QoS Performances of Multimedia Applications. In: Proc. of IEEE INFOCOM 2000, Tel-Aviv, Israel (2000)
37. Westerinen A.: Policy Terminology. In: IETF http://www.ietf.org/internet-drafts/draft-ietf-policy-terminology-02.txt (2001)
38. Wies R.: Policy Definition and Classification: Aspects, Criteria, and example. In: Proc. of DSOM'94 (1994)
39. Wies R.: Using a Classification of Management Policies for Policy Specification and Policy Transformation. In: Proc. of ISINM '95, Santa Barbara, California (1995)