Id: udo-iecon2005.tex 72 2005-07-05 08:48:17Z pohl

Model-driven Security Management of Embedded Service Systems

Stefan Illner, Andre Pohl, and Heiko Krumm

University of Dortmund
Dept. of Computer Sciences, Chair IV
August-Schmidt-Straße 12, 44227 Dortmund, Germany
{stefan.illner, andre.pohl, heiko.krumm}@udo.edu
http://ls4-www.cs.uni-dortmund.de/RVS/

Abstract— **The paradigm of Service Oriented Architectures spreads throughout the domain of business software and enterprise networks. With the proposal of the *Device Profile for Web-Services* also small, less powerful embedded devices should be able to interact with services of the network infrastructure they are connected to. New challenges arise when it comes to the adaptive management of these devices. The available computing power is often too low to allow extensive runtime evaluations for automatic adaptation to new situations. Moreover when thinking of large scaled device networks the creation and management of security policies may become a complex task. In this paper we address the latter by splitting the security management task into a design-time and run-time task. At design-time the considered access control policy is graphically modeled applying the concepts of *Role Based Access Control* and the definition is aided by a modeling tool. At run-time the configurations created by this tool are the basis for the access control computations of a security service infrastructure.**

Keywords— **model-based management, access control, embedded service systems, GRBAC, automated management**

I. Introduction

Currently not only business software is increasingly adopting the paradigms of service-oriented architectures (SOA) in order to support flexibly cooperating application systems, but also embedded real-time systems start to profit from that approach. As pointed out in [1], adaptability, reconfigurability and cooperation issues plead for an open, flexible and agile environment with "plug-and-play" connectivity as it is supported by service-oriented architectures. The SIRENA-project [2] successfully demonstrated that even the Web-Service interface paradigm can be adapted to embedded devices by means of the Device Profile for Web-Services (DPWS). Nevertheless, the flexibility and adaptation potential of SOA-based systems can only be exploited under integration of automated technical management and administration functions. However, possible solutions are faced with the memory and processor restrictions of the embedded devices. Consequently, on one hand subtle management functions are necessary which have to be designed under careful consideration of the special objectives and conditions present in the given system environment, and on the other hand the management functions can not rely on complex self-management run-time processes.

Our approach therefore aims to a light-weight, but functionally powerful run-time management. For that purpose it concentrates on a sophisticated design-phase which anticipates and analyzes all crucial run-time situations, evaluates them with respect to the existing management policies, and finally accomplishes the design of suitable and efficiently implementable run-time management functions. Due to the central role of the design-phase, we have to provide special support for the system identification, policy definition, automated system analysis and policy refinement in order to facilitate these demanding tasks and to ensure the quality of their results. The support is based on an adaptation of the model-driven approach to the domain of technical management system design, the so-called model-based management (MBM) [3], and on the provision of a corresponding modeling and design tool, the tool MoBaSeC.

MoBaSeC basically is a graphical modeling tool which supports the interactive development of object instance models by means of graphical representations, i.e. by means of object instance diagrams. The tool is accompanied by two kinds of libraries. First, class libraries correspond to domain-specific meta-models and supply predefined classes for the object instances of models. Thus, a modeler chooses a suitable meta-model and then can focus on the task of representing the embedded service system of interest by means of object and association instances. The classes contain operations which are executed by the tool and perform automated model checks and system analysis. Second, the class libraries are accompanied by rewrite system libraries. They represent graph rewriting systems [4] and consist of rule sets. Each graph rewrite rule is defined by two graph patterns, a pre-pattern and a post-pattern. Wherever the pre-pattern applies to a subgraph of the model, the rule can be applied by modifying the model in accordance with the post-pattern. The supplied rewrite system libraries correspond to management policy refinements [5]. They are used to translate the modeled abstract management policies into suitable configurations of enforcing run-time management functions, i.e., the graph rewriting enriches a given model by introducing management and security components and configuration data.

In section II, we detail the overall structure of the model in use. Afterwards, section III depicts the elements which are used in modeling environment conditions and security issues, and the relationship between these elements. It is followed by the explanation of the services providing secu-

rity related functionality and their interactions in section IV. An illustrative example is presented in section V. The explanation of the graph transformation process, by which configuration information is created automatically, follows in section VI. Section VII gives a short overview of related work, and section VIII rounds off this work.

II. Model structure

Common policy-based management approaches apply low level policies to describe management demands. In comparison to that, model-based management [6] uses very high-level policies to describe the desired behavior. Due to the utilization of a system model, the low-level policies and concrete service configurations can automatically be created by the tool. Basically a model is build up of some common and application domain specific model nodes like "Service" or "Device", and edges connecting these nodes. Each connection between two nodes expresses a specific dependency or relationship between these nodes. In common the model is divided into different abstraction layers ranging from a very abstract enterprise view, including the high-level policies to be enforced, to low-level system elements like specific hosts, devices, network protocol stacks and configurations.

The model contains three abstraction layers as shown in Fig. 1. Namely these are the *Roles & Goals* layer which contains the most abstract enterprise view, the *Services & Configurations* layer which deals with the services and abstract configuration relations between the services, and last the less abstract *Devices & File* layer containing the hardware devices, which are hosting the services, real world user credentials and the XML configurations describing the low-level management policies in form of service configuration-descriptions.

The vertical adjustment divides the model into the System, Control, Policy, and Environment sections. The *System* section contains all model elements which represent parts of the real world managed system, e.g. application services like a file- or mail-server, hosting devices, users and associated credentials. The *Control* section contains the required policy enforcement services such as the management and security infrastructure services and devices which host these services. It does not contain elements on the upper layer, as the policy enforcing components are not visible above the Services & Configurations layer. Management and security goals are modeled using specific policy objects. These are arranged in the *Policy* section of the model and are refined in the lower layers. The most right *Environment* column contains the objects required to model the environmental awareness of given management policies.

Based on this comprehensive model, our tool is able to automatically create low-level policies from the modeled high-level policies.

III. RBAC & GRBAC

This section introduces the concepts of *Role Based Access Control* and *Generalized Role Based Access Control*. We use the GRBAC model to facilitate the modeling of

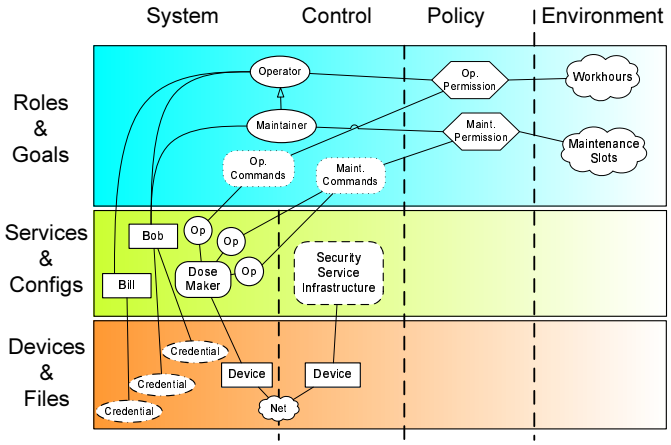


Fig. 1. Model Overview

environment-aware policies as these are required for our automatic security adaptation approach.

The *RBAC* [7] approach uses roles instead of a per user access control policy to model access permissions. These roles reflect the structure of an organization and carry specific access rights. Users or applications acting on behalf of a user are referred to as subjects in the RBAC terminology. Subjects are assigned a set of authorized roles, i.e. each subject can only act in a role that is defined in this set. In addition the RBAC model defines so called objects and operations. Objects are the entities protected by an RBAC system, e.g. services or files. Operations describe sequences of accesses of subjects acting in specific roles to RBAC objects. A formal definition of the RBAC model follows:

Subject S	a user representation in the system
Role R	a categorization primitive for subjects
Object O	a system resource
Operation T	a sequence of one or more accesses to one or more objects
AR(S)	the authorized role set for subject S
AT(R)	the authorized operation set for Role R
exec(S, T)	true iff subject S is authorized to execute operation T; true iff $\exists\, role\ R\ :\ R \in AR(S), T \in AT(R)$

Based on the active set of subject roles, the operation and the associated object a system based on the RBAC model decides whether or not the access should be granted.

As we deal with a service oriented architecture, elements of type object will not show up in our models because they are completely encapsulated by services and are accessed only internally by the services.

Role inheritance can be utilized when hierarchic structures of access rights are existing. A role R' can extend another role R, meaning that every member of R is also a member of R'.

The *Generalized Role Based Access Control* model [8] extends the RBAC model outlined above by adding two more types of roles: *object-* and *environment-roles*. Object-roles are intended to model the type or internal state of an RBAC object. They are unused in the current modeling for the aforementioned reason. Environment-roles are used to

model the environment of an application. This can be the time of day or the weight of the subject trying to access a specific object. Due to the addition of two more roles, the decision process to determine whether or not an operation should be allowed is more complicated. This process is as follows:

1. It is checked, if there exists an object-role R_O of object O, and
2. an environment-role R_E which is active at the moment, and
3. an operation T which is allowed for a subject-role R_S to access an object O in the role R_O while the role R_E of the environment is active.

Accessing an object is then replaced by performing a service call.

The upper layer in Fig. 1 exemplifies the application of the RBAC/GRBAC approach for the modeling of high-level policies.

The *Subject Roles* (e.g. Operator, Maintainer) are used as a grouping mechanism for users. A user is assigned membership in a particular subject role by connecting her to the role element. Role inheritance is expressed by directed edges between subject role nodes. *Subject Types* are used to be able to distinct clearly between the set of authorized roles and the set of roles currently occupied by a subject. As subject role membership is used in access control, there must be a *Credential* allowing the user to authenticate herself as member of a subject role. There are several types of credentials, e.g. passwords and X.509 certificates. A credential must have exactly one connection to each of subject type and user. This way, a user can have several credentials, associated with different sets of subject roles.

An *Access* (e.g. Operator or Maintainer Commands) element can be connected to concrete services and to actions of concrete services. With this construction, actions from several services can be grouped together easily in order to assign common access rights for them. When a connection between a concrete service element and an abstract service element is established, this means that all actions from the concrete service become assigned to the access.

The *Environment Roles* are abstract representatives for environmental situations (e.g. Work hours, Maintenance slots). The roles are refined to *Environment Condition* elements which describe the desired environmental situation in more detail using boolean expressions with environment condition elements (e.g. time slot definitions) connected to `and`, `or`, and `not` elements.

Connecting an environment role to a subject role has the meaning that members of this role can act as such only when the environment condition evaluates to `true`.

A *Permission* (e.g. Operator or Maintainer Permission) is the link between accesses and roles. This expresses the fact that every user being currently in at least one of the connected roles, is allowed to execute any of the actions the access element is connected to. Optionally an environment condition can be connected to a permission, implying that this permission element is used only for access computation when the environment condition evaluates to `true`.

IV. Runtime Enforcement

So far we have outlined the representation of abstract access control policies by means of graphical modeling using the MoBaSeC tool. The automated creation of the low level policies concludes the design phase of the security management process. At runtime, the infrastructure of dedicated security services provides for the enforcement of the created policies.

A. Data Structures

An XML-encoded low level policy contains several different data structures which reflect the high level elements from the graphical model. The main data structures used by the security services are the `ACL`, the `ACLEntry` and `Role` elements. An `ACL` is a simple list of `ACLEntry` elements and an additional `version` attribute which changes whenever the `ACL` or any contained sub element is altered:

```
<Acl version="0xf45eb3ab">
  <AclEntry />
  ...
</Acl>
```

An `ACLEntry` element contains several sub elements. The `Permission` element contains information about which resource is handled by this entry and if this permission is a positive one (grants access) or negative one (denies access). The `Validity` element can be used to define time related constraints for an `ACLEntry`. These time constraints are expressed through `NotValidBefore` and `NotValidAfter` elements. The `EnvCondition` element contains a logical structure over environment condition elements and defines the environmental state when the `ACLEntry` is valid. Finally the subject addressed by the entry can be specified using two different ways. On the one hand a `Subject` referred to by its public key hash may be listed explicitly in the entry. On the other hand a `RoleIdSet` containing the identifiers of the required subject roles can be supplied.

```
<AclEntry id="0x45af20e">
  <Permission />
  <Validity />
  <EnvCondition />
  <Subject /> | <RoleIdSet />
</AclEntry>
```

The `EnvCondition` elements are used beyond the security management domain also in the area of the automated management of other FCAPS areas. Each `EnvCondition` defines a logical expression over environment condition elements which describe represent low level environment data. The logical expression may include the logical operators `and`, `or` and `not`, connecting `ElementId` elements which contain the references to the environment condition elements defined somewhere else. In addition each `EnvCondition` element contains an `id` attribute which defines a unique identifier for each `EnvCondition`.

```
<EnvCondition id="0x1234567">
  <or>
    <and>
      <not>
       <ElementId />
      </not>
      <ElementId />
```

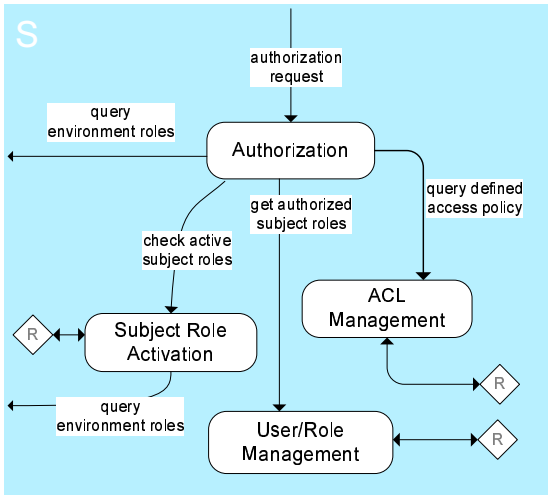


Fig. 2. Security Services

```
    ...
  </and>
  <ElementId />
  ...
 </or>
</EnvCondition>
```

A `Role` element defines a subject role used in a security context. Each role contains a `RoleId` element which defines a unique role identifier and can be used to refer to a particular subject role. In common this id consists of a describing name for a role, i.e. `MyDomain.BackupAdmin`. Additionally each role may contain an optional `MaxCardinality` element, which defines the maximum number of subjects that may act in the same role at the same time. Finally each role may contain an `EnvCondition` element which can be used to define special environmental premises that constraint the activation state of a subject role.

```
<Role>
  <RoleId />
  <MaxCardinality />
  <EnvironmentCondition />
</Role>
```

B. Security Services

The infrastructure presented here is based on four services which cooperate with each other to compute reliable access control decisions based on the prior defined access control policy. Fig. 2 depicts these services and their interaction. This infrastructure is designed to work in tight conjunction with the common management services infrastructure we already presented in [9], [10].

Each service that requires access control may forward an incoming request to the *Authorization Service*. The provided information includes the URL of the action that was called and the hash of the public key the calling user presented to the service. The authorization service now sends a query consisting of the service action URL to the *ACL Management Service* which stores the access control lists. All ACL entries are bound to complete services or specific service actions. If there exists an entry matching the given action URL, this entry is returned to the authorization service. As an ACL entry's validity can be bound to a simple time constraint, this constraint is checked first. All entries that do not pass this validity check are not further processed. In the second step the authorization service queries the *User/Role Management Service* for the set of authorized roles for the calling user. This set is checked against the required roles specified in the ACL entry and if a required role is missing, the authorization service drops this entry and continues with the next one, if available. Finally the validity of the entries concerning special environment conditions is checked. To accomplish this task the authorization service sends a query to an associated environment role activation service to check whether the involved roles are active or inactive. If the environment condition finally evaluates to `false` the processing of this particular ACL entry is aborted, continuing with the next entry, if available. In the case that no valid entry is available at last a negative access decision is returned and the called service should reject the user's request.

In the other case the authorization service now has a list of valid ACL entries, each containing a set of required roles. Now the authorization service queries the *Subject Role Activation Service* to get the set of active subject roles the user is acting in. As a subject role itself may also be constrained by additional environment conditions the subject role activation service also queries the environment role activation service to determine the activation state of the subject roles. The set of active roles is returned to the authorization service which in turn relates these to the ones defined in the ACL entries until an entry explicitly allows or denies an access or there are no entries left.

V. Example

The model presented in Fig. 3 was created with our modeling tool. It describes the situation that we encounter a dose-maker device offering a service which in turn has three actions. The `MakeDose` action is the dedicated operation of the dose-maker. As such, all operators of the owning company are allowed to invoke this action, but unauthorized employees, e.g. members of the accounting department, are not. The permission is restricted and extends to the working hours only.

The other two actions, `ManualOpenTrap` and `Configure Dose`, may only be invoked by higher-skilled maintenance personnel. Assigning authorization is always done connecting roles and accesses to permission nodes. In this example, the maintenance access consists of the two maintenance actions only. The permission to use these actions is granted to all members of the maintainer role, but only during predefined maintenance slots in order not to interrupt the regular operation of the machine.

Besides the node and edge classes required to create a specific system model, the meta-model contains special backend functions. These functions are automatically detected by the tool and made accessible via corresponding menu entries. Amongst these functions are those responsible for generating XML-based configuration files, and furthermore functions for detecting semantic errors which can not be detected during the modeling process.

For example the checking functions of the SIRENA meta-

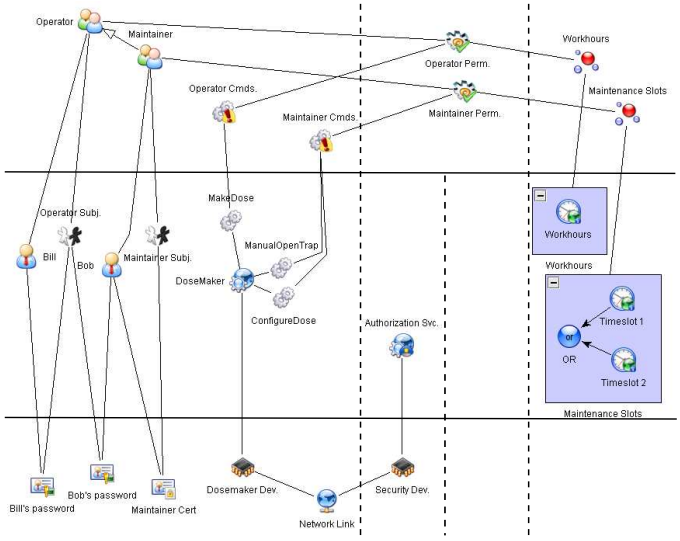


Fig. 3. Example model with access control

model include an algorithm, which checks the environment condition elements for contradictions. For the logical `and` nodes, all condition elements attached to this node are checked for their range and so it can be detected whether this condition is either always `true` or always `false`. As this will be seldom done by intention, a warning message is presented to the modeler who can then alter the model to a correct form.

VI. Graph transformation

We employ the graph transformation approach [11] in our modeling tool for automated model enhancement. The idea of this is as follows: As stated above, the model is described by a graph. Therefore it is possible to define a graph transformation system consisting of single transformation rules which alter the graph in a specified way. One such rule consists of a pre-pattern L and a post-pattern R, and on detection of a L in the model, this occurrence is replaced with an instance of R, after ensuring that this step will not violate any connectivity restrictions of the affected nodes. L and R are graphs also, consisting of elements of the same meta-model as the model graph does. The rule elements can be specified using the tool in the same way the model is specified. Some nodes in L are identified with nodes in R, while others are not. The same applies to the edges in L and R. The algorithm to apply a rule r to a model graph M is roughly outlined by the following sequence:

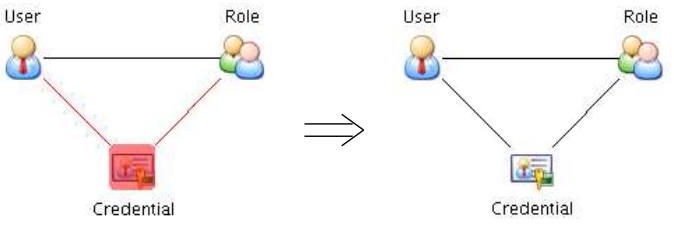
1. Find an occurrence of L in M
2. Remove all components in M which occur in L, but do not have an identity component in R
3. Insert all components into M which occur in R, but do not have an identity component in L

The application of a rule r transforms the graph G to a Graph G', denoted by $G \Rightarrow G'$. This is called a direct derivation, and G' is directly derived from G. For a better control over the application of a rule, every rule is augmented with two additional components. A *condition* allows to specify certain constraints which must be met by the detected subgraph L' in order to match the defined pattern in the search. These constraints can refer to all attribute values of elements in L'. An *effect function* is a function which is applied to the inserted graph elements, and describes how the attribute values of the inserted elements are computed from the attributes of the elements of G.

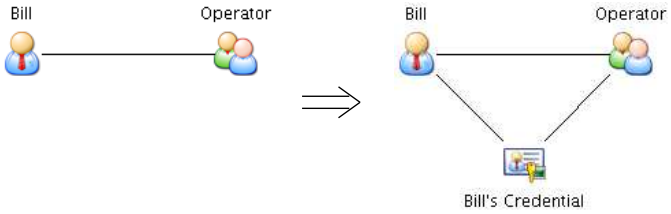
If inheritance is used in the meta-model, a rule can be made more general or more specific as needed, by using either base classes or derived classes in a rule. In order to avoid infinite recursion or other unwanted transformations, it is possible to mark pre-pattern elements as *inhibiting*, meaning that these elements must not appear in the model for the pre-pattern to match.

Applying a rule r to a Graph G several times will result in a transformation sequence $G \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n$. The result of several rule applications is called *derivation*, and G_n is derived from G.

Usually, there will not be a single rule r, but a set P of rules, where each rule serves a specific purpose. The set P is called a *graph transformation system*.



(a) A rule



(b) and its effect

Fig. 4. Graph Transformation

Fig. 4(a) shows a pre-pattern on the left and a post-pattern on the right side. The visual appearance of the credential and its connections in the pre-pattern indicates that the elements are set to inhibiting state, so that the given rule will be only applied when a user does not yet have a credential for one of her subject roles. Repeatedly applying this rule ensures that every user has a credential for every role to which she belongs, as can be seen in fig. 4(b).

The graph transformation also has another valuable benefit for model checking. It is possible to specify certain graph patterns that are allowed for the left-hand side of a rule. For each pattern it is specified whether such a pattern is mandatory or forbidden. If a mandatory pattern is given, but no matching sub-graph is found in the model, the model is invalid. A model is also invalid, if any of the forbidden patterns is detected in the model graph. This

allows the detection of semantic errors, which cannot be caught by the static analysis of the connection restrictions defined in the meta-model.

VII. Related Work

Jammes and Smit discuss the integration of SOA paradigms into the sector of the industrial automation in [1]. Their assumptions are based on the observation that emergence of powerful, networked embedded devices enables the use of higher-level communication paradigms adopted from standardized open protocol standards. This may lead to a seamless integration of device and enterprise networks.

In the area of policy creation and handling Moffett and Sloman [12] introduce the concept of a policy hierarchy by creating special low-level policies by refinement of general high-level policies.

Porto et al. [13] propose an approach extending the modeling concepts of MBM. They try to ease the creation of complex security management policies by adding an additional level of abstraction by means of so called *Abstract Subsystems* to solve problems in handling large scale models.

VIII. Conclusion

In this paper we have shown a way of integrating access control security management into our existing common management framework. The tool-assisted modeling of security policies is very similar to the modeling of common management policies. Moreover the object diagram based concept of model-based management is flexible enough to extend existing models which disregard security issues by adding the necessary elements for the desired security policy.

The presented service infrastructure easily integrates into the SIRENA service infrastructure. Each service can use the benefits of the existing authorization service by making just a simple service action call. As the computations are made outside the embedded device, even small devices can be access controlled. Nevertheless the shown concepts are currently not applicable in the domain of real-time constrained devices and services in some industrial or automotive application areas for example. The required cryptographic functions for signing SOAP messages or verifying the authentication codes of received messages do require some intense computations which would mostly break the given time constraints.

Currently we are applying our management approach to a real-world demonstrator spanning the home and automotive domain developed in the context of the SIRENA project to testify the applicability of our concepts in more complex environments.

Furthermore, research on graph grammars for transformation systems, and visual creation of complex graph transformation rules in the style of regular expressions has to be conducted.

Finally, the integration of an online deployment backend for the transmission of the generated configurations to the management services is a future goal of our work.

Acknowledgments

The work described herein was funded by the German Federal Ministry of Education and Research (BMBF) within the ITEA-SIRENA project (01ISC09G).

References

[1] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation", *Proc. of the PDCN 2005*, Innsbruck, Austria, pp. 716–723, 2005.

[2] SIRENA (Service Infrastructure for Realtime Embedded Networked Applications), `http://www.sirena-itea.org`, 2004

[3] P. Herrmann and H. Krumm, "Object-Oriented Security Analysis and Modeling", in *Proceedings of the 9th Int. Conference on Telecommunication Systems*, pages 21-32, ATSMA, IFIP, 2001.

[4] M. Andries et. al., "Graph Transformation for Specification and Programming", *Science of Computer Programming*, Vol. 34, 1999.

[5] M. Sloman, "Policy Driven Management for Distributed Systems", *Journal of Network and Systems Management*, Vol. 2, No. 4, 1994.

[6] I. Lück, C. Schäfer, and H. Krumm, "Model-based Tool-Assistance for Packet-Filter Design", in: *M. Sloman, E. Lupu, J. Lobo (Eds.), Policy 2001*, LNCS 1995, pp. 120-136, Springer-Verlag, 2001.

[7] David Ferraiolo and Richard Kuhn, " Role-Based Access Control", *Proc. of 15th National Computer Security Conference*, 1992.

[8] M. Moyer and M. Ahamad, "Generalized Role-Based Access Control", *Proc. 21st Int. Conf. on Distributed Computing Systems, Mesa, USA, pp. 391–398*, 2001.

[9] S. Illner, H. Krumm, A. Pohl, I. Lück, D. Manka, and T. Sparenberg, "Policy Controlled Automated Management of Distributed and Embedded Service Systems", *Proc. of the PDCN 2005*, Innsbruck, Austria, pp. 710–715, 2005.

[10] S. Illner, A. Pohl, H. Krumm, I. Lück, D. Manka, and T. Sparenberg, "Automated Runtime Management of Embedded Service Systems Based on Design-Time Modeling and Model Transformation", To appear in *Proc. of the INDIN'05*, Perth, Australia, 2005.

[11] M. Andries et. al.,"Graph Transformation for Specification and Programming", *Science of Computer Programming,* Vol. 34, pp. 1–54, 1999.

[12] J. D. Moffett and M. S. Sloman, "Policy hierarchies for distributed system management", *IEEE JSAC Special Issue on Network Management*, 11(9), 11 1993.

[13] J. Porto, H. Krumm and P.L. de Geus, "Policy Modeling and Refinement for Network Security Systems", *6th IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 24–33, 2005.