UML/OCL based Design and Analysis of Role-Based Access Control Policies

Oliver Hofrichter, Martin Gogolla, and Karsten Sohr

University of Bremen, Computer Science Department
Database Systems Group, D-28334 Bremen, Germany
{hofrichter,gogolla,sohr}@informatik.uni-bremen.de
http://www.db.informatik.uni-bremen.de

Abstract. Access control plays an important part in IT systems these days. Specifically Role-Based Access Control (RBAC) has been widely adopted in practice. One of the major challenges within the introduction of RBAC into an organization is the policy definition. Modeling technologies provide support by allowing to design and to validate a policy. In this work we apply a UML and OCL based domain-specific language (DSL) to design and to analyze the access control of the conference management system EasyChair. For the first time EasyChair is formally described in connection with RBAC. Our activities are located on three levels: (a) the re-engineering of the system's access control policy is located at the policy level, (b) the framework level summarizes activities concerning the RBAC metamodel (e.g. enhancements), and (c) at the configuration level, we configure a concrete policy using the conference management system options. As a result, both a DSL developed in previous work is checked for the need of enhancements, and the re-enginered EasyChair access control policy is analyzed. For validation purposes a frequently used UML/OCL validation tool is utilized.

Keywords: Metamodel, RBAC, Policy Analysis, Validation, UML, OCL

1 Introduction

Nowadays large organizations deal with a huge amount of data. Parts thereof have to be protected so that no unauthorized access can occur. At this point, access control comes into play.
Access control regulates *who* can access *what* kind of data under *which* circumstances. Different access control models exist. An access control model defines the concepts available during the creation of an access control policy [19]. An access control policy contains the concrete rules restricting access to objects in an organization. Nowadays the Role-Based Access Control (RBAC) model [16] is frequently used [13]. The idea behind RBAC is to prevent the direct assignment of permissions to users. Instead roles are interposed. In 2004 RBAC was adopted as an ANSI standard [2]. The initial RBAC model is referred to as RBAC_0 and comprises the core concepts of roles, permissions, users, sessions and relations

between these concepts. It has been enhanced by additional concepts in further versions of the RBAC model: $RBAC_1$ enhances the core RBAC model by adding role hierarchies; $RBAC_2$ introduces authorization constraints as restrictions on the RBAC functions and relations. RBAC96 cumulates the features of $RBAC_0$, $RBAC_1$ and $RBAC_2$.
Since an organization's access control policy is the basis for the decision whether access to a resource is authorized or not, it is very important that the access control policy meets the organization's security requirements. However, often access control policies are so complex that their rules get opaque. Often the formulation of authorization constraints results in additional properties the policy designer might not be aware of. That is why tool support for the design and for the validation of access control policies is desirable. In [9] a domain-specific language (DSL), based on the Unified Modeling Language (UML) [14] and the Object Constraint Language (OCL) [20], is presented which allows for designing and analyzing RBAC. For validation purposes, the UML-based specification environment (USE) [6] is applied. The organization's security administrator who faces the challenge of designing a policy for a large organization with a huge amount of resources worthy of protection can deploy the RBAC framework and in doing so is enabled to consider security requirements in early stages of software development life-cycle.
In this contribution we analyze the implemented access control policy of the EasyChair conference management and propose few minor modifications of the original RBAC DSL. EasyChair [4] was selected because using roles and role-based access control in the domain of scientific conferences seems obvious and among the conference management systems EasyChair is most commonly used. Since there was no documentation of the access control policy implemented by the EasyChair developers, role engineering had to be conducted in the first step. The EasyChair access control policy was reconstructed by (1) exploring the graphical user interface (GUI) and (2) constructing typical workflows in the work with the EasyChair system. In this way potential roles, accessible objects, access operations and authorization constraints could be mined.
The rest of this paper is organized as follows: In Section 2 the DSL based approach is introduced. In Section 3 the RBAC DSL is deployed to visualize excerpts from the re-enginered EasyChair RBAC policy. Other related work is presented in section 4. Section 5 concludes this work and commands a view on further developments in the future.

2 The Deployed DSL Based Approach

This section firstly introduces the RBAC DSL and the applied validation tool and based on this classifies the activities performed in our contribution.

2.1 RBAC DSL

The RBAC DSL allowing for the design and analysis of RBAC policies and deployed in the following is presented in [9].

It is based on a metamodel, consisting of a UML description of the core RBAC concepts and a set of authorization constraints expressed by OCL invariants. The UML part defining the abstract syntax of the DSL is depicted in Figure 1.

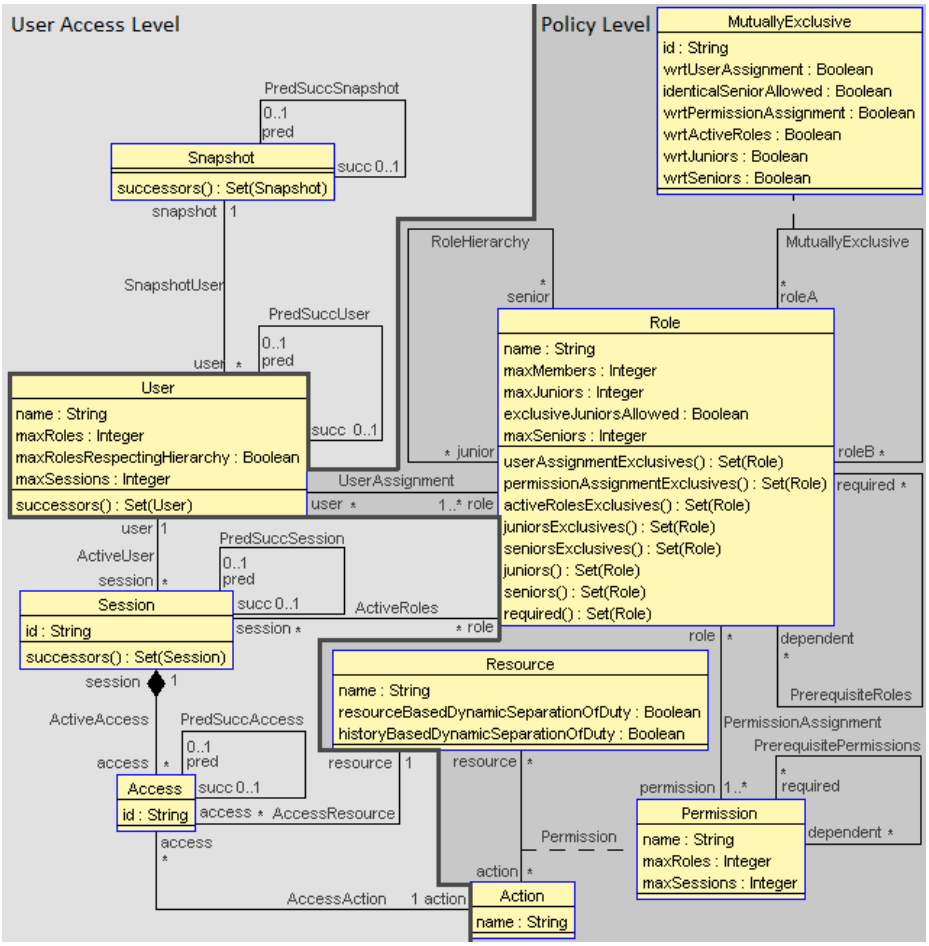

Fig. 1. Structure of RBAC metamodel

A detailed description of the OCL invariants is presented in [7]. The RBAC metamodel differentiates two levels: the dark gray shaded policy level and the light gray user access level. The policy level involves all those elements from the RBAC model the policy designer comes in contact with during the process of the design of a policy. Among these are roles, users, permissions (represented as actions on resources) and the relations between these elements such as user assignment. Similar to the elements of the user access level, these elements are represented as classes and associations. The user access level represents concrete

activities of users in the context of an access control policy. The concept of snapshot allows for the specification of dynamic constraints. A concrete access control policy is represented by an instantiation of the RBAC metamodel.

2.2 Validation Tool USE

The RBAC DSL serves as a basis for the validation. The validation is conducted by the UML-based specification environment (USE) [6]. This tool can be used to check if a policy design meets the intuition of a policy designer which is based on organizations' security requirements. In order to validate a security policy the analyst has to generate system states representing the policy. System states are represented as object diagrams. The development can be done by creation and manipulation of objects, attributes and links on a GUI or by writing command files. The created snapshots are compared with the specified RBAC model. If the modeled system states violate the defined constraints the user obtains precise feedback (cf. Figure 5) about the cause for the violation.

2.3 Analyst's Activities

This work is based on previous work [9] and extends it. The use case diagram in Figure 2 classifies the activities performed in the present work in the context of preceding activities. The activities were located on three different levels: (a) the upper part summarizes activities concerning the RBAC metamodel on the framework level. (b) activities concerning the RBAC policy which is an instance of the RBAC metamodel are located below at the policy level. (c) at the bottom of the figure the configuration level is depicted. At this level a concrete access control policy is configured. The analyst activities are promoted in the present paper (highlighted through a bold rectangle in the figure). The analyst activities span over all the three levels as we will describe in the following.
The original RBAC metamodel was developed by the RBAC metamodel development team represented as an actor in the diagram. The remaining involved actors are the EasyChair development team, the system's end-user (conference organization) and the analyst. We assume the EasyChair development team initially chose an access control model [19] at the beginning of the design phase of EasyChair. Based on this decision it defined an access control policy. It is assumed in this contribution, that EasyChair uses the RBAC model [16]. Finally the EasyChair developers defined policy configuration options for the end-users and implemented these and the according access control rules in the EasyChair system. The organizer of a scientific event as the end-user of EasyChair makes choices from the configuration options and customizes EasyChair to the needs of the respective event this way. The analyst's main activities are the policy reengineering, the policy validation and the extension of the RBAC metamodel if necessary. For the policy's definition roles, permissions and authorization constraints had to be mined. In order to mine authorization constraints the analyst also acted on the configuration level.

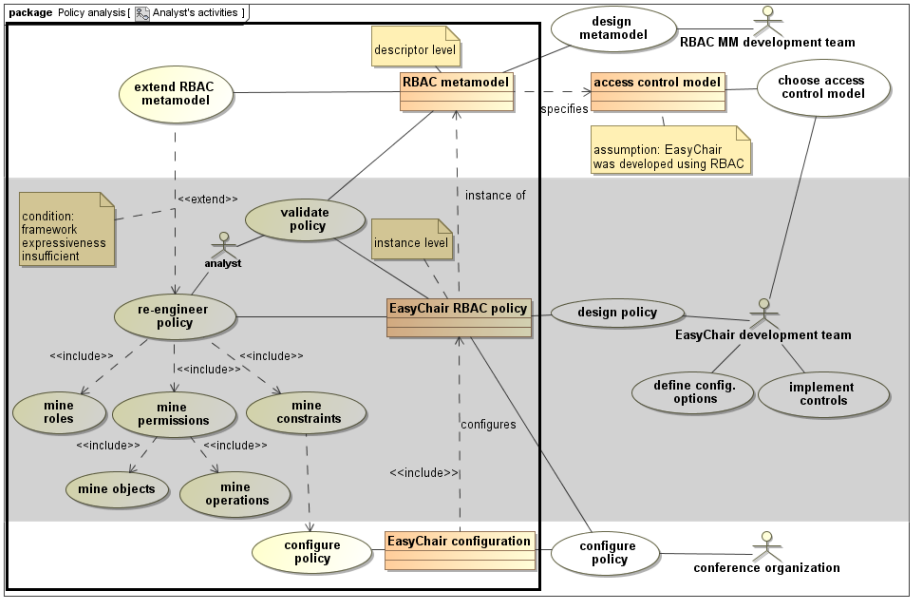

Fig. 2. Classification of analyst's activities

3 EasyChair RBAC Policy

In this section excerpts from the reverse engineered security policy of EasyChair are presented. A more detailed modeling of EasyChair's RBAC policy is presented in [7]. The following modelings serve on the one hand as example visualizations for the reverse engineered access control policy of EasyChair and describe on the other hand possibilities to validate concrete system states against a previously defined security policy. In the following figures, elements from the policy level are shaded in gray.

3.1 Modeling of the Simplest Conference

The UML object diagram in Figure 3 serves as an introductory modeling. The diagram represents the simplest of all possible academic conferences in EasyChair. The diagram depicts three users that respectively access one resource by three different actions one after another. In the first system state a user assigned to role `author` writes a paper. This paper represents the accessible resource. In the following state another user associated with the role `pcMember` reviews this paper. In the final system state a user in the role `chair` decides on the acceptance respectively the refusal of the paper. Security administrators would proceed in the same way to create organization's access control policies: by creating and manipulating objects, links and attributes.

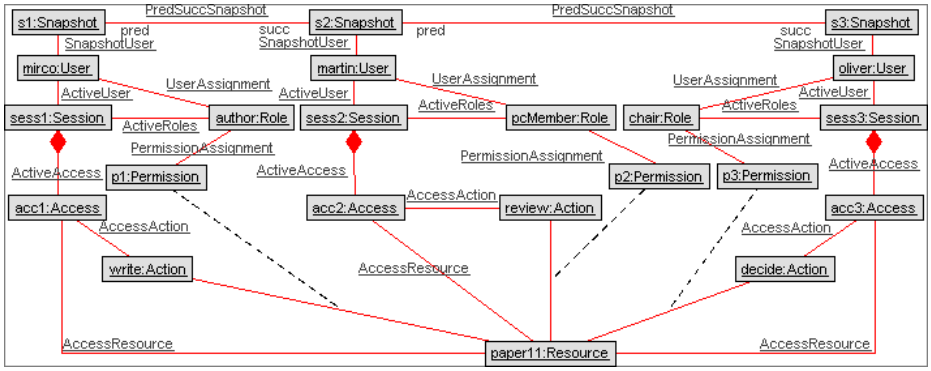

Fig. 3. Representation of the simplest academic conference in EasyChair

3.2 Policy Validation Example: Missing Permission

The previous modeling served as a visualization of an excerpt from the policy. In the following sections examples of possibilities to analyze the policy are presented.

In Figure 4, a concrete situation from the EasyChair system is depicted together with the Class invariants view of the USE tool. The situation was manually constructed on the basis of a concrete EasyChair configuration.

The policy specifies a permission for editing the administration area of EasyChair. The so-called conference configuration is represented as a resource. The user depicted in the left part of Figure 4 is associated to the role `chair`. Since this role is associated with the necessary permission for editing the conference configuration, no constraint is violated. The right part of the figure shows the access of another user to the protected resource. This user is member of the role pcMember. This role is not equipped with the needed permission. Therefore the validation tool reports on a violation of an OCL constraint depicted in Listing 1.1.

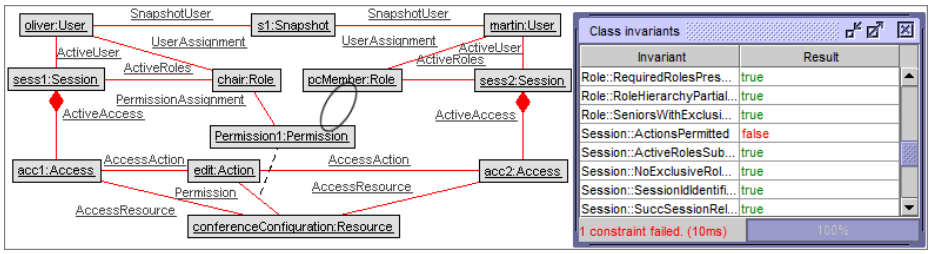

Fig. 4. Policy validation example: missing permission

Listing 1.1. Invariant `ActionsPermitted`

```
context s:Session
  inv ActionsPermitted:
    s.access->forAll(a |
      let neededPermissions = a.action.permission
          ->select(p | p.resource = a.resource) in
      neededPermissions->notEmpty() and
      s.role.permission->union(s.role.juniors().permission)->asSet()
        ->includesAll(neededPermissions))
```

The invariant `ActionsPermitted` serves as an example for the realization of OCL restrictions on the RBAC functions and relations supplementing the UML description of the RBAC concepts. The invariant in Listing 1.1 is defined in the context of an RBAC session. A session is associated with users, roles and accesses. It activates a user's membership in a role and the concrete access to a resource. As depicted in the RBAC metamodel in Figure 1, any number of accesses can be made by a user activated in a session. The invariant specifies for all the accesses that the activated roles have to be equipped with all the permissions that are needed for the execution of the respective operation (represented as action in the RBAC metamodel) on the regarded object (represented as resource in the RBAC metamodel).
Figure 5 shows an excerpt from the Evaluation Browser view in the USE tool. It enables a policy designer to identify the position where the constraint is violated. The excerpt shows that a role exists that is not equipped with all the needed permissions.

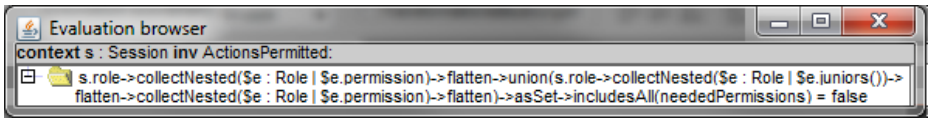

Fig. 5. Feedback concerning constraint violation

3.3 Policy Validation Example: Handling of Dynamic Constraints

Whilst the preceding modeling investigated a restriction on the static level, the following modeling serves as an example for the handling of dynamic constraints. The aim is to model the following condition: a PC member is only allowed to read the other PC member's reviews if she has already submitted her own review.
In Figure 6 a state from the EasyChair system that violates this restriction is depicted. The object diagram consists of two snapshots. In both of the snapshots a user associated to the role pcMember is regarded. In the second snapshot the regarded user accesses the other PC member's reviews. This is only permitted if there is an access to the resource through the write action first. However, this access is missing here. So the validation tool again reports on a violation of an OCL constraint responsible for the realization of this authorization constraint.

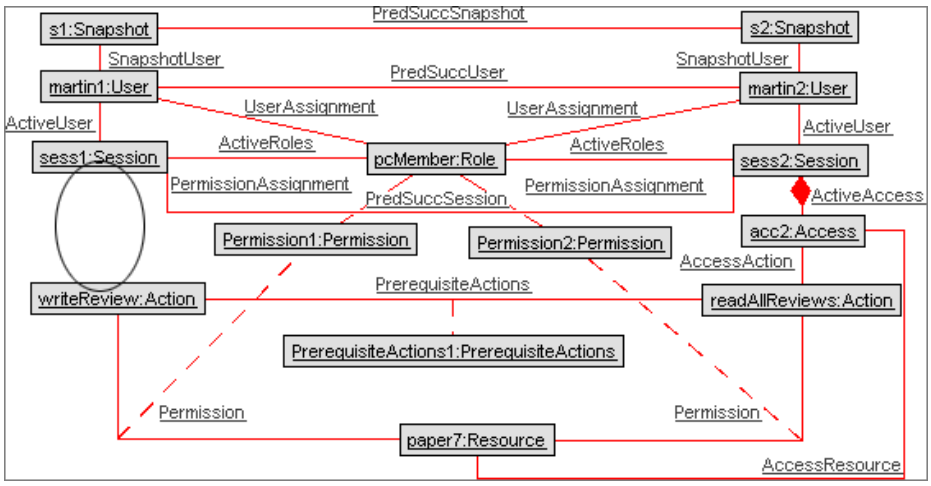

Fig. 6. Policy validation example: handling of dynamic constraints

3.4 Metamodel Extension

The extensibility of the RBAC metamodel is shown in [17]: the metamodel is extended by support of delegation and revocation concepts. In some places in this contribution the modeling of the re-engineered EasyChair RBAC policy also involved extensions to the original RBAC metamodel [9]. A detailed description of the extensions is presented in [7]. For example the metamodel was extended to prevent that reviewers have the same affiliation as authors. For the realization the classes `User` and `Action` and the association class `MutuallyExclusiveActions` were expanded. On top of this an additional invariant was introduced. The class `User` was expanded by the attribute `affiliation`. The class `Action` was expanded by an operation for identifying mutually exclusive Actions regarding the attribute `wrtAffiliation` introduced to `MutuallyExclusiveActions`. These ingredients are brought together by the OCL invariant `checkAffiliation` defined in the context of the class `Resource`.
In the same way the RBAC metamodel can be extended if an organization's policy designer is confronted with situations that can be expressed by the existing metamodel.

4 Related Work

A classification of RBAC related publications since the adoption of role theory in information security [16] is presented in [5]. Three major classes of RBAC research were identified in [15]. In the present paper, a hybrid approach for the definition of roles was applied. Other approaches are top-down [12] and bottom-up (also labeled as role mining). The RBAC framework consists of a family of models [16]. Each family member represents an access control model. In the

present contribution a concrete access control policy is designed and analyzed. The relationship between access control models and access control policies is described in [19]. Interoperability problems of access control policies are addressed by the standard access control policy language XACML [21]. Juerjens combined Model-driven development and security by integrating security related information in UML [8]. Different approaches for modeling RBAC properties in UML/OCL exist. In [10] a UML profile for a modeling environment is presented. The approach uses (customized) class diagrams and activity diagrams. For verification of RBAC properties, OCL is applied. In [18] UML is also used to describe security policies. Contrary to our approach the UML models are transformed to Alloy for analysis purposes. An access control policy analysis approach for mobile applications using the UPPAAL model checker is presented in [1]. A survey of Model-driven security offers [3].

5 Conclusion and Future Work

In this contribution, we have successfully applied a UML and OCL based DSL for the investigation of a concrete access control policy. In doing so, the DSL itself was validated and described how the DSL in combination with a UML and OCL validation tool can be employed to discover and eliminate undesired policy properties which do not meet the security requirements.
The policy design and validation were carried out by the same group of persons. In order to validate concrete system states against a previously defined policy, a separation of the responsibilities as in software quality management [11] is of prime importance. Since the conducted role mining based on the exploration of the GUI, simulation of concrete work steps in EasyChair and interviews with domain experts the policy designed for investigation already based on findings of the investigation. This is why in this work the RBAC DSL was primary applied to formalize and visualize the reengineered RBAC policy of EasyChair.
For the future, an automatic transformation from concrete RBAC policies into concrete syntax used by RBAC DSL (USE) would be desirable. To harmonize the different application-specific access control policy languages the language XACML was developed. A further step could be to develop a transformation of the XACML representation of a policy to the USE syntax. The other direction, namely the transformation from RBAC policy formulated in RBAC DSL into a representation supported by concrete target systems like Tivoli or DirXMeta-Role, could be of interest as well.
Moreover, the usability of the applied approach has to be improved. For example at the moment the policy level and the user access level have to be modeled in the same diagram. Separate diagrams and syntactic representations of (our DSL) modeling elements would improve usability within an enterprise.
Another interesting aspect that has to be considered in future work is the evolution of the RBAC metamodel. This comprises among other things the development of a concept for handling different metamodel versions and both domain-specific metamodel changes and ones concerning access control model concepts.

References

1. Abdunabi, R., Ray, I., France, R.B.: Specification and analysis of access control policies for mobile applications. In: Conti, M., Vaidya, J., Schaad, A. (eds.) SACMAT. pp. 173–184. ACM (2013)
2. ANSI: Role Based Access Control (2004), ANSI/INCITS 359-2004
3. Basin, D.A., Clavel, M., Egea, M.: A decade of model-driven security. In: Breu, R., Crampton, J., Lobo, J. (eds.) SACMAT. pp. 1–10. ACM (2011)
4. EasyChair Conference System. Internet, `http://www.easychair.org/`
5. Fuchs, L., Pernul, G., Sandhu, R.S.: Roles in information security - A survey and classification of the research area. Computers & Security 30(8), 748–769 (2011)
6. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. SCP 69, 27–34 (2007)
7. Hofrichter, O.: Analyse und Modellierung der rollenbasierten Zugriffskontrolle für ein IT-System zur Verwaltung wissenschaftlicher Konferenzen. Master's thesis, University of Bremen (2012)
8. Jürjens, J.: UMLsec: Extending UML for Secure Systems Development. In: Jézéquel, J.M., Hußmann, H., Cook, S. (eds.) UML. LNCS, vol. 2460, pp. 412–425. Springer (2002)
9. Kuhlmann, M., Sohr, K., Gogolla, M.: Comprehensive Two-Level Analysis of Static and Dynamic RBAC Constraints with UML and OCL . In: Baik, J., Massacci, F., Zulkernine, M. (eds.) Proc. SSIRI. pp. 108–117. IEEE (2011)
10. Montrieux, L., Wermelinger, M., Yu, Y.: Tool support for UML-based specification and verification of role-based access control properties. In: Gyimóthy, T., Zeller, A. (eds.) SIGSOFT FSE. pp. 456–459. ACM (2011)
11. Myers, G.J.: Art of Software Testing. John Wiley & Sons, Inc., New York, NY, USA (1979)
12. Neumann, G., Strembeck, M.: A scenario-driven role engineering process for functional RBAC roles. In: SACMAT. pp. 33–42 (2002)
13. O'Connor, A.C., Loomis, R.J.: 2010 Economic Analysis of Role-Based Access Control. Tech. Rep. RTI Project Number 0211876, NIST (2010)
14. OMG (ed.): UML Superstructure 2.4.1. OMG (Aug 2011)
15. Sandhu, R.S.: Future Directions in Role-Based Access Control Models. In: Gorodetski, V.I., Skormin, V.A., Popyack, L.J. (eds.) MMM-ACNS. LNCS, vol. 2052, pp. 22–26. Springer (2001)
16. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. IEEE Computer 29(2), 38–47 (1996)
17. Sohr, K., Kuhlmann, M., Gogolla, M., Hu, H., Ahn, G.J.: Comprehensive Two-Level Analysis of Role-Based Delegation and Revocation Policies with UML and OCL. Information and Software Technology 54(12), 1396–1417 (2012)
18. Sun, W., France, R.B., Ray, I.: Rigorous Analysis of UML Access Control Policy Models. In: POLICY. pp. 9–16. IEEE Computer Society (2011)
19. di Vimercati, S.D.C.: Access Control Policies, Models, and Mechanisms. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security (2nd Ed.), pp. 13–14. Springer (2011)
20. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Object Technology Series, Addison-Wesley, Reading/MA (2003)
21. OASIS XACML. Internet, `http://docs.oasis-open.org/xacml/`