

Generation of Role-based Access Control Requirements from UML diagrams

Manuel Koch, Karl Pauls
Freie Universität Berlin
Berlin, Germany
{mkoch,pauls}@inf.fu-berlin.de

Francesco Parisi-Presicce
George Mason University
Fairfax, VA-USA
fparisi@ise.gmu.edu

Abstract—Security is a crucial aspect in any modern software system. Yet, security requirements engineering is not sufficiently supported in the software development process. We present an approach to the generation of role-based access control requirements from UML diagrams and a systematic refinement of these requirements. The refinement is driven by access control constraints which are integrated automatically into the access control requirement model. Our approach uses only notions available in UML to ensure integration into the UML software development process.

I. INTRODUCTION

Security is an integral part of most modern software systems that are not used in completely trusted environments, but is not yet sufficiently supported in the software development process. The lack of a systematic support for software engineers who need to produce secure software is based on the fact that security requirements are generally difficult to analyze and model [15], [5], and that security policies are generally specified in terms of highly specialized security models which are not integrated with general software engineering models. Recent research concerns the integration of security engineering into the software development process [9], [8], [1], [14].

In this paper we present an approach to the development of role-based access control [19] policies for distributed systems with an example of view-based access control (VBAC) [2]. Our approach generates access control requirements from UML use case and UML sequence diagrams, part of the UML software engineering process for functional requirements. To ensure an integration into the UML software development process, we present the generated access control requirements into a UML class diagram, which requires no particular security expertise on the part of the system engineer. Due to a notion well known to system engineers and the generation of requirements, system engineers consider access control early in the software development process and the specification of access control requirements becomes less error-prone.

A second contribution of this paper is the systematic refinement of the generated UML access control requirement diagram. The refinement is driven by access control constraints, which describe, in a declarative way, access control requirements that have to be satisfied in any system configuration. We follow the approach of Ray et al. in [17] in which access control constraints are specified by UML object diagrams representing forbidden object states.

The UML diagram for the access control requirements is refined if it does not ensure the satisfaction of the access control constraints. To check this, we model the dynamic behaviour of the system operations by UML object diagrams representing the pre- and post-conditions of the operations. The UML notion for access control constraints and operation pre- and post-conditions again simplifies the integration into the UML development process. Furthermore, we believe that the visual representation of access control constraints and operation pre- and post-conditions is more legible and easier to understand for software engineers. On the other hand, the graphical notion of access control constraints and system operations allows us to adapt to our purposes a checking algorithm developed in the theory of graph transformations [7]. This algorithm detects operations that may construct object states violating constraints and adds additional pre-conditions to operations which ensure satisfaction. Finally, the generated pre-conditions are transferred to the UML diagram for access control requirements.

The generation and refinement process accompanies the UML software development process. If new sequence diagrams for functional requirements are added or further refined, or access control constraints are modified, a new generation and subsequent refinement become necessary. Hence, the final access control specification must be developed along the usual software development process.

We summarize the main contributions of the paper:

- Generation of role-based access control requirements from UML sequence diagrams resulting in a UML diagram.
- Visual specification of access control constraints and the dynamic behaviour of system operations by UML object diagrams.
- A systematic way to refine access control requirements on the basis of access control constraints.
- Integration of all steps into the UML software development process.

The remainder of the article is organized as follows: Section II introduces View-based Access Control used as running example. Section III presents the generation of access control requirements from UML use case and sequence diagrams. Section IV concerns the refinement of the UML diagram

for the access control requirements. Section V compares our approach to related work and Section VI concludes the article and points to future work.

II. VIEW-BASED ACCESS CONTROL

View-based access control (VBAC) is an access control model specifically defined to support the design and management of access control policies in object-oriented systems [4], [2]. VBAC extends role-based access control [19] by *views* as an additional layer between roles and permissions. A view is a grouping concept of fine-grained access rights, which are permissions to call operations of distributed objects. Views are assigned to roles and a subject can call an operation if it has a role with a view that contains the permission required to call the operation.

In the sequel, we exemplify the VBAC model with a small conference management application [4], [2] used also in the rest of this article. In the conference management system, the program committee chair (PC) can issue a call for papers to open a submission phase for a conference, so that authors may submit papers. The PC is responsible for the declaration of the submission deadline, which terminates the submission phase and starts the reviewing phase. The PC assigns reviewers to the papers. The reviewing phase is terminated by the PC, who calls for a final decision. A possible class diagram for this application is shown in Figure 1.

A VBAC policy consists of a set of *roles* and a set of *views*. In the example, we have the roles Chair, Reviewer and Author.

```
policy Conference {
  roles
  Chair
  Reviewer
  Author ...}
```

There may be several views in a VBAC policy. Views are defined with the keyword **view**, the view name and the list of permissions to call operations. One view defines only permissions for operations with respect to one class, i.e., a view does not have permissions for operations belonging to different classes. One example view is shown below: *Paper-View* gives the permission to call the operations *write()* and *createReview()* of the class *Paper*.

```
view PaperView {
  write(text:String)
  createReview(reviewerID:int):Review}
```

In the sequel, we show how to systematically develop the access control requirements to specify a VBAC policy.

III. ACCESS CONTROL REQUIREMENTS IN UML DIAGRAMS

The specification of the VBAC policy for a given application is a difficult design task, especially since system designer are usually not security experts. Therefore, the designer should be supported in the software development process to obtain the access control requirements and their translation into a VBAC

policy. In this section we present a methodology to develop the access control requirements:

- 1) Generate roles from use case diagrams (Sec. III-A), and views from sequence diagrams (Sec. III-B).
- 2) Model the behavior of (security relevant) operations by object diagrams (Sec. IV-A).
- 3) Model access control constraints declaratively by object diagrams (Sec. IV-B).
- 4) Check automatically which operations may violate which constraints (Sec. IV-B)
- 5) Add pre-conditions to the operations to ensure constraint satisfaction (Sec. IV-C).

The process accompanies the software development process and parts must be executed several times, e.g., if sequence diagrams are added or further refined in the functional software development process. Also new access control constraints may trigger a new process execution. Therefore, the final access control specification is developed along the usual software development process. Our aim is an access control requirement process that does not require security expertise and uses UML diagrams well known to system designers.

A. Generating access control roles from use case diagrams

We briefly review the generation of access control roles from UML use case diagrams as presented in [3], where access control roles correspond to UML actors. Therefore, from the use case diagram of the conference management example in Fig. 2, we derive the access control roles Chair, Reviewer and Author.

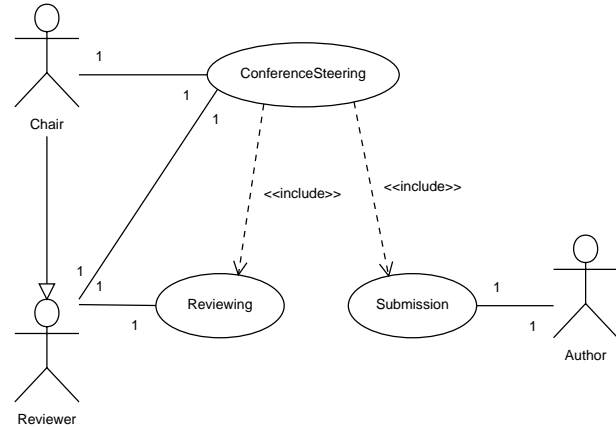


Fig. 2. The use case for the conference application.

B. Generating views from UML sequence diagrams

This section concerns the generation of access control requirements from sequence diagrams. Sequence diagrams contain inherent access control requirements since they specify the required accesses of actors to call operations in order to fulfil their functional tasks. Therefore, the actors (access control roles) need the permissions to call all the operations

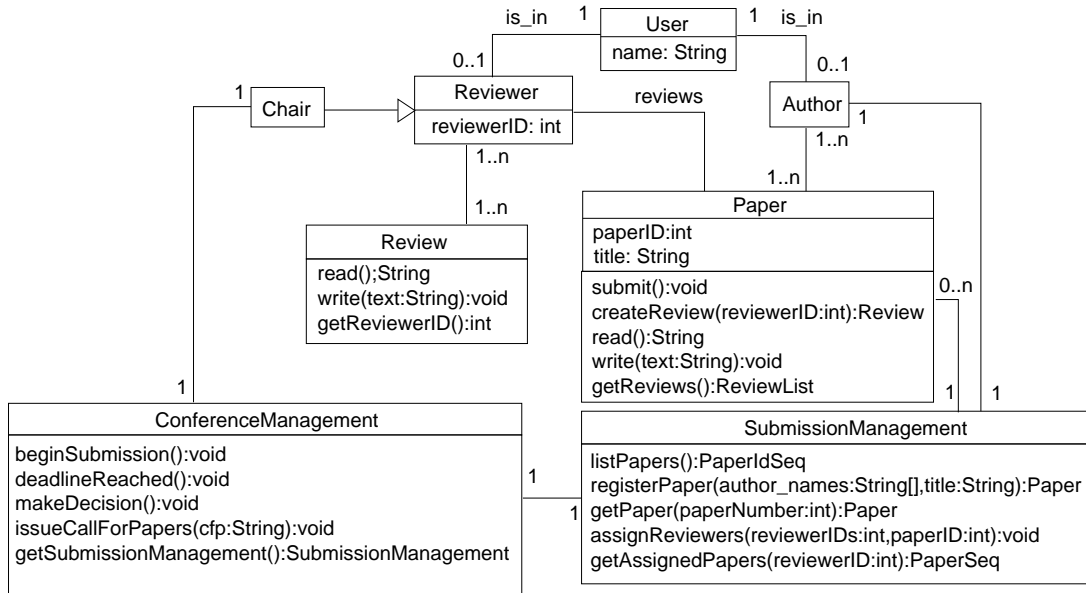


Fig. 1. The class diagram for the conference application.

used in the sequence diagrams. Figures 3, 4 and 5 are sequence diagrams for our example. In the first one, an author must be able to call the operation *getSubmissionManagement()* on *ConferenceManagement*, the operation *registerPaper()* on *SubmissionManagement* and the operations *write()* and *submit()* on *Paper*.

The generation of access control views from a sequence diagram *S* can be described as follows:

A view is generated for each object in *S* on which at least one operation is called. The permissions of the view are all the operation calls on this object.

From the sequence diagram in Fig. 3, a view is generated for the objects *paper:Paper*, *cm:ConferenceManagement* and *sm:SubmissionManagement*. There is no view on the object *author:Author* since no operation is called on this object. On the object *paper:Paper*, there are the operation calls *write()* and *submit()*, giving rise to a view which contains the two access rights *write* and *submit* (see Fig.6). On the conference management object *cm* there is only the call *getSubmissionManagement()*. This gives a view (called *ConfMgmt2* in Fig.6) consisting of only one permission, *getSubmissionManagement*. Since the operation is also called by the reviewer in the sequence diagram in Fig. 5, there is also a connection between the *Reviewer* and the view *ConfMgmt2* in Fig. 6. The class *SubmissionManagement* has a view containing the permission *registerPaper*, since this operation is called by the author in the sequence diagram. Other views are similarly generated for the sequence diagrams Fig. 4 and Fig. 5. All the generated views are shown in the diagram in Fig. 6, which we call the *view diagram*. Since sequence diagrams usually specify only scenarios, the generated view diagram does not contain the complete access control information and it must be refined. Section IV presents the refinement on the basis of access control constraints.

The view diagram also contains the access control roles, generated from UML use case diagrams, which are assigned to views by associations. There is an association between a role and a view if the view contains operation permissions generated from a sequence diagram in which the actor has called all the operations in the view.

IV. REFINEMENT OF ACCESS CONTROL REQUIREMENTS

The previous section presents the generation of views from sequence diagrams. The generated views contain the permissions needed to fulfil the functional tasks of the roles. It is possible, however, that the assigned permissions are too powerful, allowing a role to call an operation in situations in which such a call shall not be allowed. This may be based on domain specific information not considered in the UML diagrams. Therefore, the access control model must be restricted and the view diagram refined. The restriction of the access control model is specified by declarative *access control constraints*. Access control constraints describe security requirements that have to be satisfied in any system configuration. The following requirements are access control constraints for our running example:

- 1) The chair must not submit a paper to his/her own conference.
- 2) An author must not review her/his submitted paper.

A designer uses the access control constraints to express, in a declarative way, the access control requirements (s)he wants to ensure.

This section presents a methodology to support the designer in the constraint-driven refinement of the generated view diagram so that the resulting access control model permits only accesses satisfying the constraints. This methodology is divided into two steps:

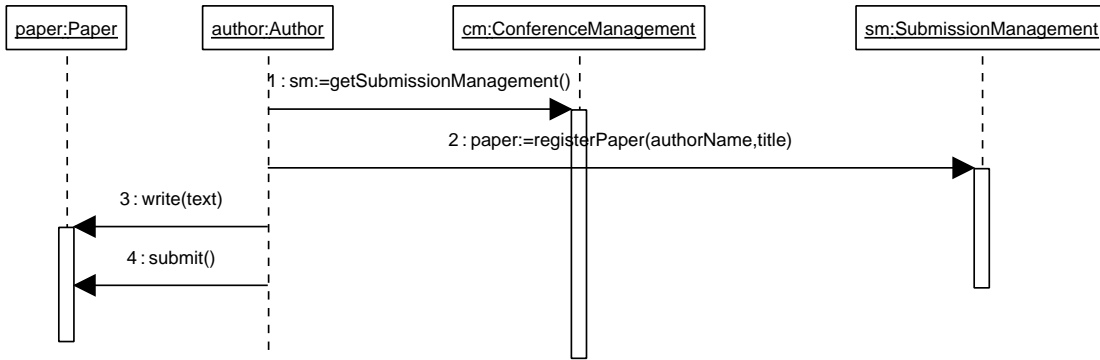


Fig. 3. The sequence diagram for the Author's view.

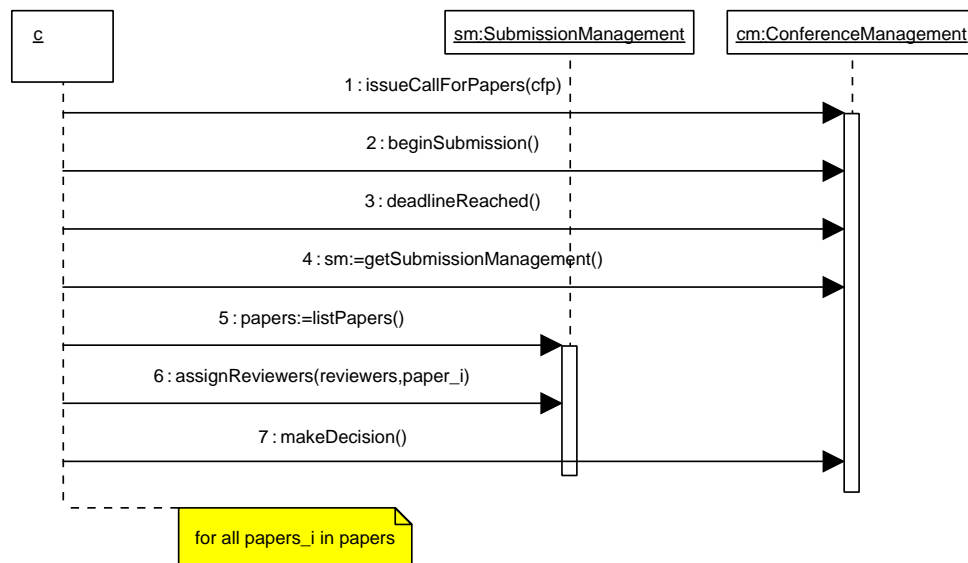


Fig. 4. The sequence diagram for the Chair's view.

- 1) Check if the access control constraints can be violated based on the views in the view diagram.
- 2) If the access rights in the view diagram are not sufficient to ensure the satisfaction of the constraints, modify the view diagram.

We show next how these two steps can be performed automatically, using an existing theoretical result in the field of *graph transformations* [18], [10], [12]. To apply the results, we specify the dynamic behaviour of the system operations with object diagrams in the next subsection.

A. Modelling the dynamic system behaviour

Each operation in the class diagram is specified by an object diagram which specifies the pre- and post-conditions of the operation. We use the stereotype `<<create>>` to distinguish between pre- and post-state. The object state before the execution of the operation consists of all the objects and

links without the stereotype `<<create>>`, the object state after the execution of the operation consists of all the objects and links in the diagram. This means that the operation creates the objects and links with the stereotype `<<create>>`¹. Consider, as an example, the operations `assignReviewers()` and `registerPaper()` in Fig. 7. In the object diagram for `assignReviewers()`, only the link between the *reviewer* object and the *paper* object carries the stereotype `<<create>>`. Therefore, the operation creates only this link, while the remaining object structure must already exist to call the operation. The intended meaning of the operation is that a user in the role *reviewer* is assigned to review the paper. In the object diagram for `registerPaper()`, the *paper* object is created together with the links to the *author* and *submgmt* object. The operation registers a new paper of the author to

¹Operations may destroy objects, as well. In this case, another stereotype must be introduced. For the sake of space, however, we omit this case here.

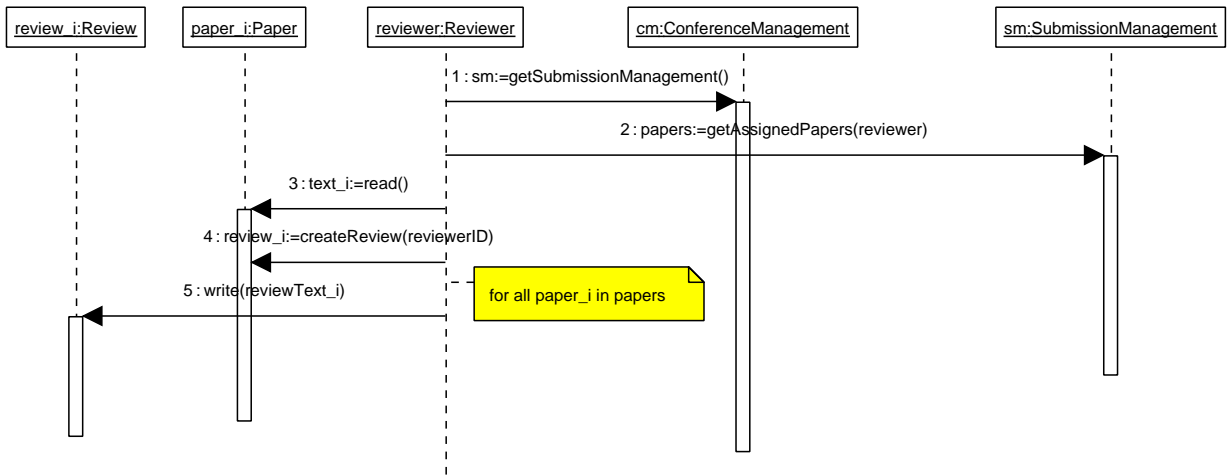


Fig. 5. The sequence diagram for the Reviewer's view.

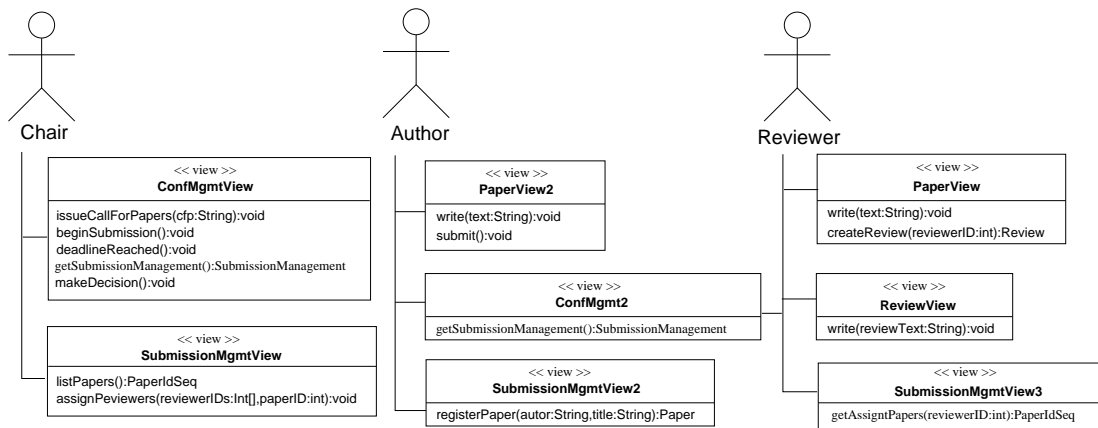


Fig. 6. The generated access control views.

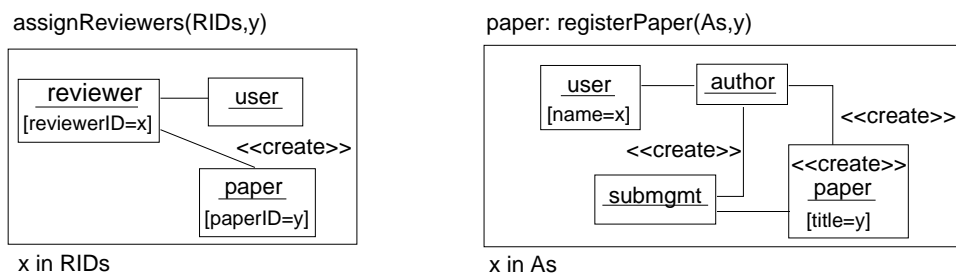


Fig. 7. Object diagrams for pre- and post conditions for operations.

the conference.

The object diagrams for the operations describe the dynamic behavior of the system. System states can be generated by *applying* the object diagrams to the current system state. An operation object diagram *OD* can be applied to an object diagram *S* (representing the current system state) if the object structure of the operation object diagram *OD* without the stereotype `<<create>>` can be found as sub-state in *S*. Then, all the `<<create>>` objects and links of *OD* are added to *S* to generate the new system state. For example, the operation diagram `assignReviewers()` in Fig. 7 can be applied to a system state *S* if *S* contains a reviewer object and a user object which are connected. If this is the case, the application adds a paper object to *S* with a connection to the reviewer object.

B. Checking the access control model

To check, automatically, if the operations can construct a system state which violates the access control constraints, we translate the constraints into object diagrams as well. Ray et al. [17] present the specification of access control constraints by object diagrams. In their approach, a constraint is an object diagram which represents an invalid sub-state. An object state satisfies a constraint if the object diagram of the constraint does not occur as a sub-state. The object diagrams for our two examples are shown in Fig. 8. Diagram 1) specifies a state in which a user is in both roles chair and author and (s)he has a paper submitted to the conference for that (s)he is chair. Diagram 2) shows the state in which a user is in both the role author and the role reviewer, and there is a paper of the user which (s)he reviews. These constraint sub-states must not occur in any system state to ensure constraint satisfaction.

It is necessary to check if the object diagrams for the operations can construct an object state which violates a constraint object diagram, i.e. an object state in which a constraint diagram occurs as a sub-state. The checking algorithm is based on a theoretical result presented in [7], where it is developed for arbitrary graphs. Since object diagrams can be easily seen as graphs, the results can be applied to our case. We do not explain the algorithm in detail here, but only show the ideas and the result with respect to our example. The interested reader is referred to [7], [11]. The idea of the algorithm is to check for each operation object diagram and each constraint object diagram, to determine whether the operation object diagram can create an object state which contains the constraint object diagram. Informally, this is the case if the operation diagram creates object structures which are part of the constraint diagram. If we consider our example diagrams in Fig. 7 and Fig. 8, the operation `assignReviewers()` may violate constraint 2) by assigning a user to review his/her own paper. The operation `registerPaper()` may violate constraint 1) by registering to the conference a paper of the chair. The operation `assignReviewers()`, however, cannot violate constraint 1) since it does not create any structure occurring in the constraint object diagram. Therefore, the algorithm determines that these two operations can violate the constraints, and thus the access control model must be

refined to ensure constraint satisfaction. How this refinement modification is discussed in the next subsection.

C. Refinement of the access control model

When the algorithm detects which operation can violate which constraint, the call of the violating operations is restricted by an additional pre-condition, so that they can be called only if they construct valid object states. The algorithm generates these additional pre-conditions, as well (see [7], [11]). We use these pre-conditions and integrate them into the operation object diagrams by extending the object diagrams by a stereotype `<<not>>`. The intended meaning of an operation object diagram with this stereotype is that the operation can be called in a certain object state if all objects and links without `<<not>>` and `<<create>>` stereotype occur as a sub-state in the object state, but this sub-state does not contain all the objects and links with stereotype `<<not>>`.

Figure 9 shows the modified operations of the example. The object diagram for the operation `assignReviewers()` is extended with a pre-condition which prevents the user from being the author of the paper that needs a review. The pre-condition for the object diagram of `registerPaper()` prevents the operation call if the user is the chair of the conference to which the paper is submitted.

Finally, the pre-conditions of the operations in the object diagrams are transferred to the UML view diagram. The pre-conditions are added to the operations into an OCL-like notation and have the following schema: **cond** *variables opPos* \Rightarrow **not** (*opNeg*). Both **cond** and **not** are keywords; *variables* defines the variables used in the expressions *opPos* and *opNeg*, respectively; the expression *opPos* specifies the object structure of the operation object diagram which does not carry the stereotypes `<<create>>` or `<<not>>`; the expression *opNeg* specifies the object structure of the operation object diagram which does not carry the stereotype `<<create>>`. The two operation object diagrams in Fig. 9 are transformed into the UML view diagram in Fig. 10.

V. RELATED WORK

Work related to our approach to security engineering is presented in [1]. Basin et al. describe a model driven approach [6], called SecureUML, to develop role-based access control policies for J2EE applications. A formal basis allows the designer to reason about access control properties, and tool support is given by an integration of SecureUML into the ArcStyler tool [16]. Compared to our approach, however, the analysis stage of the software process is not considered and the process starts with the design models.

Jürjens [9] presents the integration of security into the UML and shows how to model several security aspects by UML model elements such as, for example, stereotypes or tagged values. His approach is more general than ours, since it is not restricted to access control, but considers, for example, security protocols. Furthermore, Jürjens considers a wider variety of UML diagrams (e.g. also deployment diagrams) since he is concerned with the integration of security into the

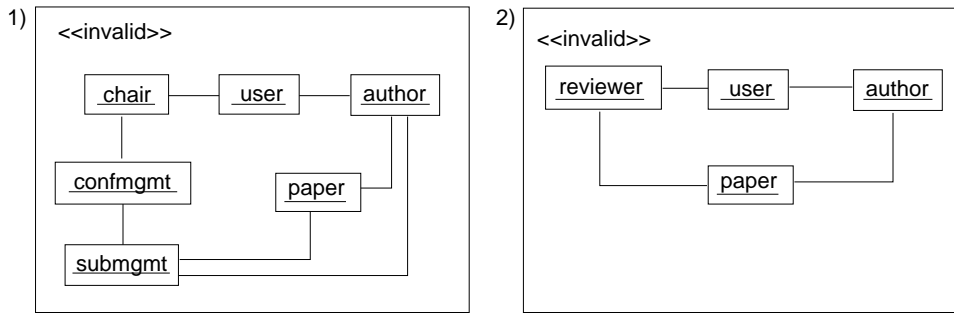


Fig. 8. Object diagrams for access control constraints.

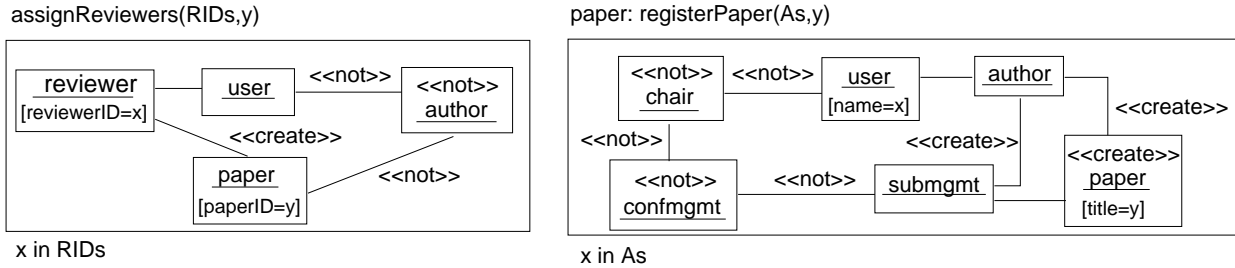


Fig. 9. Object diagrams for the modified operation object diagrams.

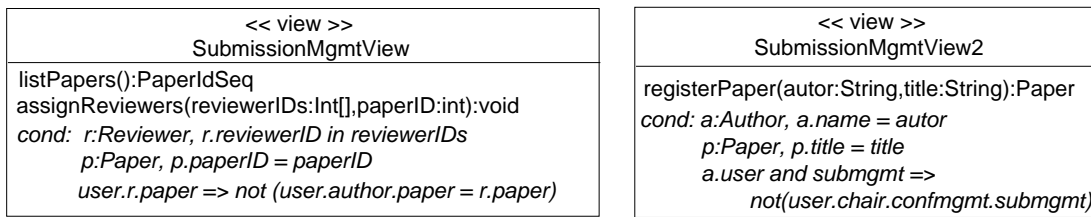


Fig. 10. Pre-conditions for the operations in the UML view diagram.

UML in general. Our approach focuses on access control and the generation and enforcement of access control policies in distributed systems, and only the UML diagrams necessary to fulfil our aim are used.

VI. CONCLUSION

This article discusses the integration of access control requirements engineering into the UML software development process. We have presented the generation of role-based access control requirements from UML use case and UML sequence diagrams. The generated access control requirements are represented in UML class diagrams. We have also presented a systematic way to refine access control requirements with respect to additional access control constraints.

Although not described here for lack of space, rules and constraints have a well-defined semantics based on graph transformations [13].

Future work will investigate the integration of our approach into a model-driven approach and into the Eclipse project, including the generation of the access control model from UML diagrams.

REFERENCES

- [1] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security. *Engineering Theories of Software Intensive Systems*, 2004.
- [2] G. Brose. Manageable Access Control for CORBA. *Journal of Computer Security*, 4:301–337, 2002.
- [3] G. Brose, M. Koch, and K.-P.Löhr. Integrating Access Control Design into the Software Development Process. In *Proc. of 6th International Conference on Integrated Design and Process Technology (IDPT)*, 2002.
- [4] Gerald Brose. *Access Control Management in Distributed Object Systems*. PhD thesis, Freie Universität Berlin, 2001.
- [5] Premkumar T. Devanbu and Stuart Stubblebine. Software engineering for security: A roadmap. In Anthony Finkelstein, editor, *The Future of Software Engineering*. ACM Press, 2000.
- [6] D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley and Sons., 2003.
- [7] R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars - a constructive approach. In *Proc. SEGRAGRA'95 Graph Rewriting and Computation*. Electronic Notes of TCS, 1995. <http://www.elsevier.nl/locate/entcs/volume2.html>.
- [8] J. Juerjens. UMLsec: Extending UML for Secure Systems Development. In *Proc. of 5th Int. Conf. on the Unified Modeling Language*, Lect. Notes in Comp. Sci. 2460, pages 412–425, Springer, 2002.
- [9] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005.
- [10] M. Koch, L.V. Mancini, and F. Parisi-Presicce. Conflict Detection and Resolution in Access Control Specifications. In M.Nielsen and U.Engberg, editors, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS)*, Lect. Notes in Comp. Sci. 2303, pages 223–237. Springer 2002.

- [11] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A Graph Based Formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):332–365, August 2002.
- [12] M. Koch, L.V. Mancini, and F. Parisi-Presicce. Decidability of Safety in Graph-based Models for Access Control. In M.Waidner D.Gollmann, G.Karjoth, editor, *Proc. of 7th European Symposium on Research in Computer Security (ESORICS)*, Lect. Notes in Comp. Sci. 2502, pages 229–243, Springer 2002.
- [13] M. Koch, and F. Parisi-Presicce. UML Specification of Access Control Policies and their Formal Verification. *Software System Modeling (SoSyM)*, to appear 2005.
- [14] T. Lodderstedt, D. Basin, and J. Doser. SecureUML:A UML-Based Modeling Language for Model-Driven Security. In *Proc. of 5th Int. Conf. on the Unified Modeling Language*, Lect. Notes in Comp. Sci. 2460, Springer, 2002.
- [15] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In Anthony Finkelstein, editor, *The Future of Software Engineering*. ACM Press, 2000.
- [16] Interactive Objects. Arcstyler, 2005. www.io-software.com.
- [17] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to Visualize Role-Based Access Control Constraints. In *Proc. of SACMAT'04*, pages 115–124. ACM, 2004.
- [18] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. World Scientific, 1997.
- [19] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. In *Proc. of the 5th ACM Workshop on Role-Based Access Control*. ACM, July 2000.