Tool Support for UML-Based Specification and Verification of Role-Based Access Control Properties

Lionel Montrieux Michel Wermelinger Yijun Yu
Centre for Research in Computing & Computing Department
The Open University, Milton Keynes, UK
L.M.C.Montrieux@open.ac.uk M.A.Wermelinger@open.ac.uk Y.Yu@open.ac.uk

ABSTRACT

It has been argued that security perspectives, of which access control is one, should be taken into account as early as possible in the software development process. Towards that goal, we present in this paper a tool supporting our modelling approach to specify and verify access control in accordance to the NIST standard Role-Based Access Control (RBAC). RBAC is centred on mapping users to their roles in an organisation, to make access control permissions easier to set and maintain.

Our modelling approach uses only standard UML mechanisms, like metamodels and OCL constraints, and improves on existing approaches in various ways: designers don't have to learn new languages or adopt new tools or methodologies; user-role and role-permission assignments can be specified separately to be reused across models; access control is specified over class and activity diagrams, including 'anti-scenarios'; access control is automatically verified. The tool is built on top of an existing modelling IDE and allows for automatic verification of models according to our RBAC modelling approach, while providing users with the ability to easily identify and correct errors in the model when they are detected.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques — *Computer-Aided Software Engineering*

General Terms: Security, Verification.

Keywords: Security, UML, RBAC, model, verification, OCL, Model-Driven Engineering.

1. INTRODUCTION

Security is a growing concern in the software engineering community [8]. As software systems are increasingly connected and handle more and more sensitive data, making sure that access is restricted to only those who are authorised becomes a crucial concern. Giving users access to too much data or processes will increase the risk of misuse, while not giving users the permissions they actually need will prevent them from getting their work done.

Too often is security dealt with at the end of a software development project, leading to ill-designed systems with poor protection of data and other assets. As Fernandez-Medina et al. point out [4], security would greatly benefit from being taken into account as early as possible in the software development process, for example as part of a Model-Driven Engineering process. This would make it easier to: keep track of security requirements and to make sure that the security measures that have been selected actually enforce those requirements; handle change and its impact on security; communicate the security measures to the stakeholders who do not have a computer science background and are not able or not willing to read code.

This paper presents tool support for modelling RBAC configurations and properties on several UML diagrams, and for verifying that the model actually does enforce those access control properties. The tool is a part of a modelling IDE, allowing users to use it in a familiar environment, together with their usual modelling activities.

The paper is organised as follows: we present background and related work in Section 2, then introduce the tool and the approach it supports in Section 3. We discuss future work in Section 4 and conclude in Section 5.

2. BACKGROUND

In this section, we present the RBAC standard, as well as existing Model-Driven Security approaches.

2.1 RBAC

RBAC is an access control model that has been standardised by the NIST [9]. The main idea behind RBAC is to map users to roles and roles to permissions, instead of mapping users to permissions directly. Roles are supposed to match the users' actual roles in an organisation. Since roles, and not users, are assigned permissions, it makes access control configuration easier to maintain within an organisation: when a new person joins, the administrator only has to assign her to the roles she needs to get her work done. If that person's responsibilities within the organisation change later, then it's just a matter of adding and/or removing roles. If a role's responsibilities change, then changing the role's permissions accordingly will be reflected in what all the users assigned to that role can and cannot do.

The RBAC standard is divided in four different levels, each level adding new functionalities on top of the previous one. Level 0 defines user-role assignments as well as

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

Level	Features
0	users, roles, permissions
1	role hierarchies
2	constraints (Separation of Duty)
3	review

Table 1: RBAC features for each level

role-permission assignments. Each user can be assigned any number of roles, and each role can be assigned any number of permissions. Level 1 adds role hierarchies: these are partial orders in which roles can inherit other roles' permissions. Level 1a, also called General Hierarchical RBAC, supports arbitrary partial orders, while level 1b, also called Limited Hierarchical RBAC, comes with limited partial orders. The limitation depends on the vendor's choice. Next, level 2 adds Separation of Duty (SoD) constraints: to prevent conflicts of interest, it is possible to forbid a user to be granted two roles at the same time. This enforcement can be done either at the user-role assignment level, using static constraints, or within a session, using dynamic constraints. Finally, level 3 adds a permission-role assignment review requirement: it must be possible to review which roles are assigned to a specific permission, and which permissions are assigned to a specific role. Table 2.1 summarises each level's requirements.

2.2 Model-Driven Engineering

Model-Driven Engineering (MDE) is the software engineering approach that is built around the principle that *everything is a model*. Models are defined according to a metamodel, but the growing number of metamodels led to another higher level to describe metamodels: meta-metamodels [3]. Several organisations and companies propose platforms to support MDE. One of them, arguably the most widely used, is the OMG's Model-Driven Architecture approach [10], which includes UML (Unified Modeling Language) models, OCL (Object Constraint Language) constraints and MOF (Meta-Object Facility) metamodels and meta-metamodels.

In the security world, Fernandez-Medina et al. point out that "*current approaches which take security into consideration from the early stages of software development do not take advantage of Model-Driven Development*"[4], but it is a direction that is being developed, including by Basin et al. [2], who define Model-Driven Security (MDS) as a specialisation of MDE, where "*a designer builds a system model along with security requirements, and automatically generates from this a complete, configured security infrastructure*".

2.3 Modelling RBAC properties on UML

Several approaches exist for modelling RBAC properties on UML. UMLsec [6] uses UML's extension mechanisms to express security properties. It is not limited to access control, but its support of RBAC is limited, as it does not support all the levels of the standard. RBAC properties can be defined on activity diagrams only. Tool support exists to verify models against the RBAC properties.

Another approach, SecureUML [7], uses class diagrams instead of activity diagrams to express access control configurations and properties. Classes are used to model users and roles, and association classes give permissions. SecureUML can also generate EJB code enforcing the RBAC properties.

Finally, authUML [1] allows one to define RBAC properties on use case diagrams, which are usually used early in the development cycle. authUML modelling is done in three consecutive steps: first, defining and processing access control requirements, by transforming the requirements into predicates; second, ensuring consistency, completeness and conflict-free accesses of Use Cases; third, ensuring consistency, completeness and conflict-free accesses for operations. We are not aware of any tool support for authUML.

3. OUR APPROACH

Our approach differs from the other ones in that it allows one to model RBAC configurations and properties on several UML diagrams. We use the standard UML extension mechanisms to provide extra annotation capabilities to existing UML diagrams: class diagrams, activity diagrams, and access control diagrams, that are a custom version of class diagrams to model users, roles, permissions, and their assignments.

It is also different from other approaches by including anti-scenarios: while other approaches focus only on verifying that a specific user has access to a specific resource, we also allow one to model anti-scenarios, where a specific user *must not* have access to a specific resource.

3.1 RBAC profile

The access control configuration can be seen on the access control diagram: users, roles and permissions are represented using stereotyped classes. Associations between users and roles and between roles and permissions are the assignments. The entire RBAC standard is supported: role hierarchies can be represented using class generalisation, and SoD constraints can be expressed using OCL constraints.

The class diagram allows one to stereotype operations whose access should be restricted. These stereotypes must come with at least one association to a permission in the access control diagram. The set of associations represents the permissions needed to perform the operation.

Finally, the activity diagram is the place where the access control properties are defined: stereotyped partitions represent users, and those stereotypes must come with an association with a user in the access control diagram. Stereotyped partitions also come with associations to roles: these are the roles that the user represented by the partition has activated before the activity starts. This set of roles must necessarily be a subset of the roles assigned to the user. Then, actions in the diagram can also be stereotyped. An action that a user needs to be able to perform is stereotyped with «`granted`», while an anti-scenario, i.e. an action that a user can never be able to perform is stereotyped with «`forbidden`». Both these stereotypes come with associations to operations in the class diagram, that describe the operations involved in the action. Actions can also be stereotyped with «`activateRoles`» (resp. «`deactivateRoles`») to express that a set of roles is activated (resp. deactivated) before the action starts, and deactivated (resp. re-activated) right after it finishes.

Figure 1 is a sample model annotated using our approach. It represents a system allowing professors and teaching assistants to fill a database with students' marks. The students are only able to read their marks. There is a hierarchy relationship between professors and teaching assistants so that users assigned with the Professor role inherit from the per-

missions assigned to the Teaching Assistant role. The model is made of one access control diagram, one class diagram and one activity diagram. In the activity diagram, the user Doe starts the activity with no role activated, and activates the Student role to perform an action. This is to demonstrate the possibility to activate roles for a specific action. Associations between stereotypes are normally not visible on the diagrams, but we have represented them as notes for the reader's convenience.

3.2 Consistency

Because we use several types of diagrams, it is essential to ensure that they do not contradict each other. To enforce consistency between diagrams, we have defined a set of 20 OCL constraints. For example, one of them makes sure that an action stereotyped with «`granted`» is not also stereotyped with «`forbidden`»: it would not make sense to have an action that a user must at the same time *always* be able to perform and *never* be able to perform.

SoD constraints can be added by the user to the profile. While static SoD ensures that the same user can not be assigned both of two roles, dynamic SoD only enforces that these two roles can not be activated in the same time. Static SoD is verified on the access control diagram, by checking the user-role assignments while taking role hierarchies into account. Dynamic SoD is verified on activity diagrams, by making sure that the two conflicting roles are not activated in the same time on a single action.

3.3 Verification

The verification of RBAC properties is done using two similar OCL contraints: one for actions stereotyped with «`granted`», and another one for actions stereotyped with «`forbidden`». In both cases, we compute two sets: the set $Perm_{needed}$ of permissions required for the action to be performed, and the set $Perm_{activated}$ of permissions that the user has activated when s/he tries to perform the action. For actions stereotyped with «`granted`», the verification succeedes if the user has activated at least all the permissions required, i.e. if $Perm_{needed} \subseteq Perm_{activated}$. For actions stereotyped with «`forbidden`», the verifications suceeds if the user has not activated all the required permissions, i.e. if $Perm_{needed} \not\subseteq Perm_{activated}$.

3.4 Implementation

Our approach has been implemented as a UML profile for IBM Rational Software Architect [5]. The profile, together with a few sample models, are available online[1].

During the modelling process, the model will go through stages where it violates one or several constraints. Live evaluation, i.e. evaluation of the constraints every time a change is performed to the model, is therefore not suitable for this approach, as it would trigger a lot of unnecessary errors. The OCL constraints are therefore verified in batch mode. The tool then issues errors and warning that indicate which OCL constraint has been violated and where, both in the console and in the diagram, as one can see in Figure 2. The verification of the example in Figure 1 takes about 3 seconds on a computer with a 2.53GHz Intel i5 M 460 CPU and 4GB of RAM, using Rational Software Architect version 8.0.1

Our RBAC profile can simply be added to an existing modelling project, and it is possible to use it immediately.

[1]http://computing-research.open.ac.uk/rbac

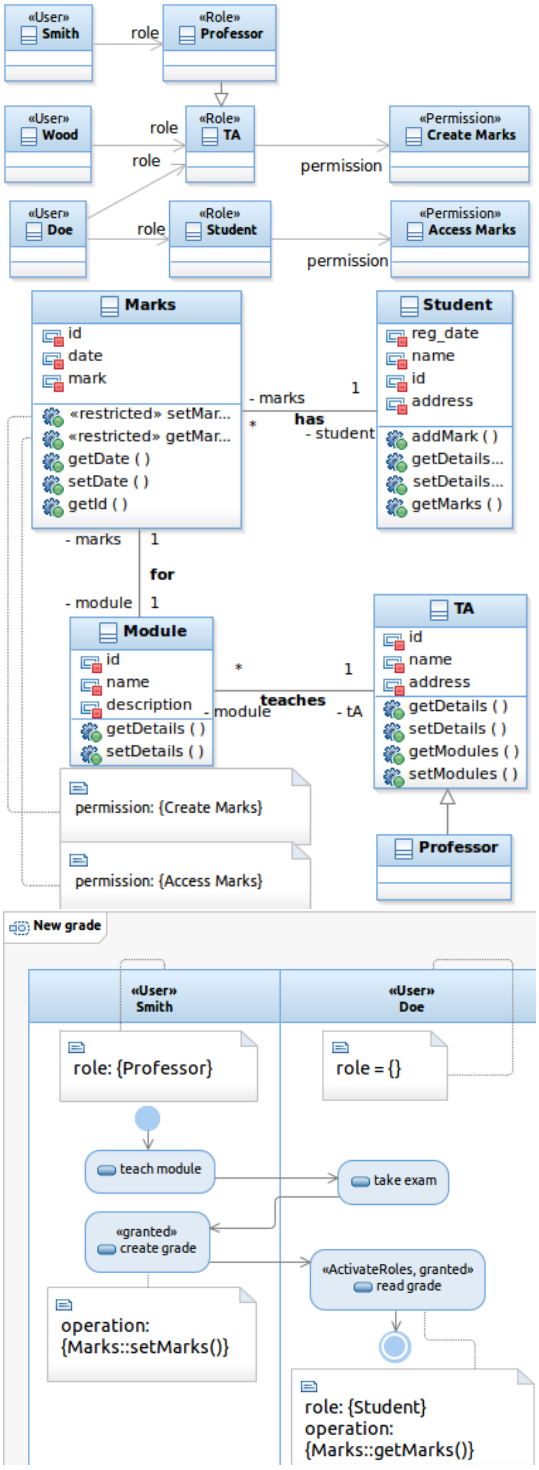

Figure 1: A sample model of a students' mark system

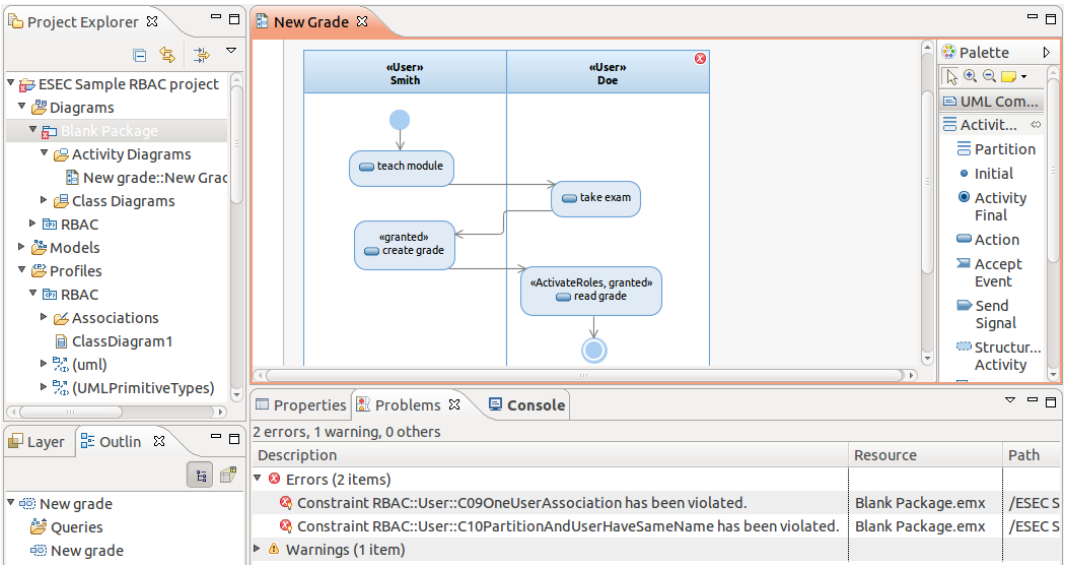

Figure 2: Screenshot of the tool showing errors in the model

The OCL validation command will then verify the annotated model against the RBAC properties specified on the activity diagrams. If errors are found, they are reported both on the graphical representation of the model and in an error message. The combination of both notifications makes it easy to identify where a problem lies, and what constraint the model violates. The tool stays out of the way of the user, allowing her to proceed with her modelling activities, especially those not directly related to access control concerns.

4. FUTURE WORK

Future releases of the software will extend the metamodel and support model evolution.

The extension of our approach's metamodel will allow us to support more UML diagrams, such as sequence diagrams. Each type of diagram provides a different view on the software, and it is important to be able to represent access control in each of the views in which it makes sense to do so.

Software evolution is the other direction we will follow to improve our tool, both for step-by-step evolution and for software merging. After a sequence of changes, only the OCL rules whose evaluation may have been impacted should be re-checked. Repair actions will then be suggested when a model is found insecure, in order to fix it.

Merge situations typically arise when two organisations become one. They will eventually have to merge their access control configurations and properties. We will provide support to help users during the merge operations, by detecting potential conflicts as well as similar elements that might be merged, and by providing repair actions to make the resulting model satisfy the merged properties.

5. CONCLUSION

We presented a UML profile for Rational Software Architect that allows one to model RBAC configurations and properties on a UML model, using three different types of diagrams. The profile contains OCL constraints that ensure that the RBAC-related annotations are consistent, and will raise both textual and visual errors when they are violated. OCL constraints can also be used to enforce static and dynamic SoD constraints. The verification of the RBAC properties expressed on activity diagrams is performed with two OCL constraints that make use of the annotations in the three supported types of diagrams to make sure that the annotated model does enforce the properties expressed. Although the profile has been developed for use with Rational Software Architect, the fact that we only use standard technologies makes it easy to port to other modelling IDEs that support modelling of UML models, extensions of the UML metamodel, and verification of OCL constraints.

6. REFERENCES

[1] K. Alghathbar and D. Wijesekera. authUML: a three-phased framework to analyze access control specifications in use cases. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, FMSE '03, pages 77–86, 2003.

[2] D. A. Basin, J. Doser, and T. Lodderstedt. Model driven security for process-oriented systems. In *SACMAT*, pages 100–109. ACM, 2003.

[3] J. Bézivin, F. Jouault, and D. Touzet. Principles, Standards and Tools for Model Engineering. In *ICECCS*, pages 28–29. IEEE Computer Society, 2005.

[4] E. Fernández-Medina, J. Jürjens, J. Trujillo, and S. Jajodia. Model-Driven Development for secure information systems. *Information & Software Technology*, 51(5):809–814, 2009.

[5] IBM. Rational Software Architect 8.1, 2010.

[6] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2005.

[7] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, 2002.

[8] J. Münther. On the Security of Security Software: Invited Position Paper. *Electr. Notes Theor. Comput. Sci.*, 142:5–10, 2006.

[9] R. S. Sandhu, D. F. Ferraiolo, and D. R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *ACM Workshop on Role-Based Access Control*, pages 47–63, 2000.

[10] R. Soley and the OMG staff. Model Driven Architecture. white paper, November 2000. http://www.omg.org/cgi-bin/doc?omg/00-11-05 (Last accessed 14 June 2010).