Challenges in Model-Based Evolution and Merging of Access Control Policies

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

Access Control plays a crucial part in software security, as it is responsible for making sure that users have access to the resources they need while being forbidden from accessing resources they do not need. Access control models such as Role-Based Access Control have been developed to help system administrators deal with the increasing complexity of the rules that determine whether or not a particular user should access a particular resource. These rules, as well as the users and their needs, are likely to evolve over time. In some cases, it may even be necessary to merge several access control configurations into a single one. In this position paper, we review existing research in model-based software evolution and merging, and argue the need for a specific approach for access control in order to take its specific requirements into account.

Categories and Subject Descriptors: I.6.4 [**Computing Methodologies**]: Simulation and Modeling – *Model Validation and Analysis*

General Terms: Design, Security, Verification

Keywords: Security, UML, RBAC, model, verification, evolution, incremental verification, merging, OCL, access control.

1. INTRODUCTION

Security is a growing concern in the software engineering community. Software systems are increasingly connected, and they have to manage very sensitive data, such as credit card numbers, health records or corporate secrets. Access control, specifically, is an important part of a system's security measures, as it is responsible for giving users access to the resources.

Several access control models have been developed to help administrators handle their access control configurations. Role-Based Access Control (RBAC) [21] is one of them. It introduces *roles*, which are assigned to users and give them access to permissions. RBAC is an NIST standard that is divided in 4 levels, each level adding new requirements, such as role hierarchies or separation of duty constraints, on top of the levels below. It is a widely-known model that has been used as the basis for other models, such as PRBAC [20] or OrBAC [11]. In this paper, we choose to use RBAC as an illustration of access control, as it contains many of the usual and most interesting constructs of access control models, such as roles, hierarchies and separation of duty constraints.

With rules on who can access what growing more and more complex, the maintenance of access control configurations has become a very complicated problem. Administrators struggle to find the right balance between giving users enough permissions so they can get their work done and preventing them from accessing resources they should not have access to. It is therefore essential to assist developers and system administrators in designing the right access control configurations, so they can be confident that the configuration they have chosen actually does enforce the access control properties they need to follow.

Furthermore, the access control configuration is likely to evolve, as new users come in, existing users leave or get new responsibilities and privileges. The roles and role hierarchies can also change, and so can the permissions. Particularly with complex configurations, even the smallest change may have unpredicted consequences. System administrators' work would be made much easier if they had the possibility to make sure that the changes they want to introduce will not break any of the security properties they have previously defined. In this paper, we illustrate this problem with a simple example, where a university uses a software system to keep track of students' grades.

Another case where an access control configuration will evolve is the merging of two organisations: whether it is a company buying another one or two organisations who decide to become one, they will eventually have to merge their access control configurations. Such an operation will probably be non-trivial, as conflicts will arise with the same role having completely different permissions in both organisations, redundancies in the role hierarchies or in the permissions, or conflicts in the security properties. To illustrate this case, we merge two different companies' configurations.

The rest of this paper is organised as follows: section 2 discusses model-driven engineering approaches for security, and especially for access control. Then, section 3 focuses on software evolution in general, and reviews the existing approaches for software evolution and incremental consistency checking. In section 4 we show how the general approaches

IWPSE-EVOL'11, September 5–6, 2011, Szeged, Hungary.

name	diagrams	config	levels	anti-pat.
secureUML	class	class	all	no
UMLsec	activity	tagged val.	0 - 1	no
authUML	use cases	predicates	all	no
ours	class, act.	access ctrl.	all	yes

Table 1: UML-based RBAC modelling approaches

discussed in the previous section are not optimal when it comes to access control specifically, and we give a general overview of how the problems highlighted in the previous section can be solved, and we finally conclude in section 5.

2. MODEL-DRIVEN ENGINEERING

Model-Driven Engineering (MDE) is a model-centric approach to software engineering, where typically models are constructed to provide a high-level description of a software system, and then progressively refined into more detailed models until the source code, which is also a model, is produced. Arguably the most popular and widespread MDE approach is Model-Driven Architecture (MDA), from the OMG. It includes standards such as Universal Modeling Language (UML) or Object Constraint Language (OCL).

2.1 Model-Driven Security Engineering

As the need for secure systems is growing, it becomes increasingly important to integrate security concerns as soon as possible in the software development cycle, therefore making security a « first class » concern. This is especially true for systems handling sensitive data. However, Fernandez-Medina et al. point out that approaches that take security into account early in the development cycle do not necessarily make use of MDE [7]. Model-Driven Security (MDS) is the approach pushed by Basin et al. [2] to integrate those security concerns in a wider MDE approach.

Several MDS approaches have been proposed, allowing one to model specific security concerns as well as verify the model against properties or requirements that it should fulfil. This is in line with the challenges in software modelling identified by Van Der Straeten et al. [22], such as domain-specific modelling or model validation and verification.

2.2 Modelling RBAC properties with UML

Several approaches have been developed for modelling RBAC configurations and properties on a UML model. Table 1 compares four of these approaches: secureUML [12], authUML [1], UMLsec [9] and our approach [18].

The different approaches first differ in the diagrams they support, and ours is the only one to support several types of UML diagrams. The way the access control configuration is represented is another difference. SecureUML uses classes, while UMLsec uses tagged values attached to an activity diagram, authUML uses predicates, and our approach uses an extension of class diagrams that we call access control diagrams. All approaches but UMLsec support the entire RBAC standard, and only our approach allows for the definition of anti-patterns, which are requirements on what a user should *not* be allowed to do.

3. MODEL-BASED EVOLUTION

Software is rarely something that is built once and then never updated again. Instead, during its life-cycle, it will receive not only bug fixes, but also new features, use new technologies, or interact with different software packages, in order to comply to changing requirements. Mens et al. [17] identify several challenges in software evolution. The challenges we focus on in this paper are supporting model evolution, in order to support evolution of high-level artifacts such as models, formal support for evolution, that leads to the development of formalisms to help software evolution, and integrating change in the software life-cycle, in order to make it usable by practitioners as an integrated part of their daily activities.

In the Model-Driven Engineering terminology, evolution and fixes to models are *transformations*. Mens et al. [16] give a taxonomy of model transformations that helps identifying exactly what kind of transformations should be considered in the case of model-driven evolution of access control properties. Since the taxonomy targets graph transformation tools, we do not review all the categories presented, but instead select a few of them that will help us understand what kind of transformation we are dealing with: the number of source and target models, endogenous and exogenous transformations, horizontal and vertical transformations, as well as syntactic and semantic transformations.

A first criteria is the number of source and target models. While in the case of evolution of an access control configuration, the transformation is one-to-one, in the case of merging several configurations into one, it is a many-to-one transformation. In both cases, the transformation is endogenous, i.e. the source and target are instances of the same metamodel, and also horizontal, as the transformations do not involve refinement or generalisation of the model. More importantly, the transformations need to be semantical, as semantic information is important to make sure that the resulting model is still valid regarding the access control properties expressed.

UML itself does not come with any support for model evolution. Fortunately, its extensible nature makes it possible to build support for model evolution on top of UML. One of the existing approaches for dealing with evolution of UML models is to use reuse contracts integrated in the UML metamodel [14]. Another approach is Judson et al.'s pattern-based transformations of UML models [8], that allows one to define transformations as metamodels, and models can be checked against those metamodels for conformance to the specified transformation. This approach, however, is limited to class and sequence diagrams. Jürjens et al. propose UMLseCh to model evolution of UMLsec properties [10]. It uses an approach similar to Mens', but focused on security.

3.1 Incremental consistency checking

Consistency checking is about making sure that a model is consistent, which means that its different diagrams do not contradict each other, and that there is no contradiction within a single diagram. Consistency checking can be divided in two parts: the detection of inconsistencies, and their resolution. In the context of software evolution, consistency checking is about making sure that a particular change to a model will not introduce any new inconsistencies. Given the scope of this paper, we focus on inconsistency detection and resolution for evolving models.

3.1.1 Inconsistency detection

The first part of the consistency checking process is to identify inconsistencies. Our approach for modelling RBAC properties and configurations [18] makes use of OCL constraints to enforce some consistency properties between different elements, whether or not they are part of different diagrams. Egyed [4] also uses OCL constraints to enforce consistency of UML models, providing almost instant verification of 24 consistency rules. As opposed to our approach, which currently does not handle evolution and therefore verifies all the consistency rules every time the verification is triggered, Egyed's approach is to only verify the constraints that access elements that have been changed during the evolution of the model. Constant monitoring of model changes allows for immediate feedback, at the price of memory usage.

Another approach is provided by Blanc et al. [3], where rules are defined using predicate logic, and checked with a Prolog engine, which also allows to detect which rules need to be re-checked when the model is modified. Yet another one is Nentwich et al.'s xlinkit [19], that performs pairwise consistency checking on any XML document, including an XMI representation of UML models.

3.1.2 Inconsistency resolution

Once the inconsistencies have been detected, they need to be resolved. Egyed provides a way of fixing inconsistencies [5], and a way of generating and evaluating the several possible choices [6].

Mens proposes an approach using graph transformations to incrementally resolve inconsistencies [15], while Nentwich et al.'s xlinkit also provides possible fixes.

During consistency resolution operations, two properties are essential: correctness and completeness. Correctness is about making sure that the proposed changes will lead to a consistent model. Completeness is about making sure that all the possible changes leading to a consistent model are generated. While the approaches discussed here all ensure correctness, their completeness is limited to atomic changes that do not introduce any new element.

3.2 Model merging

Model merging is, as we discussed earlier, a many-to-one transformation. In his survey on software merging [13], Mens provides a classification of existing software merging techniques. The particular type of merging we are interested in is the one that applies to the example we mentioned in the introduction: two organisations want to merge their access control configurations and properties. The first distinction made by Mens is between two-way and three-way merging. Three-way merging takes into account a common ancestor of the models to be merged, while two-way merging does not.

Another important factor Mens discusses is how the models are represented, which can lead to textual, syntactic, semantic or structural merging. Textual merging only considers the artefact's as text, usually dealing with it on a line-by-line basis. Syntactic merging can take the syntax of the language or model into account, producing a syntactically correct model. Semantic merging takes it a step further by also taking the semantics into account. Finally, structural merging improves on semantic merging as it also allows one to make sure that behaviour is preserved, but it has the downside that user input can sometimes be necessary to resolve a conflict that a structural merge algorithm can not solve automatically.

Finally, the type of information considered is another aspect of software merging considered by Mens: state-based techniques use information available from the current versions of the models to be merged, as opposed to change-based techniques that consider all the changes that lead to the current version. Then, operation-based techniques treat change as transformations.

Most approaches for model merging, including Mens' reuse contracts [14], focus on the merging of models that have been modified by separate developers.

4. MODEL-BASED EVOLUTION OF ACCESS CONTROL

Now that we have reviewed the existing, general model evolution approaches on UML, we discuss in this section how they are not directly suited to deal with the specificities of security, and access control in particular, as well as how solutions focused on access control could be developed.

This section presents the three steps that need to be taken in order to better handle evolution of access control at model-level, in the correct order: inconsistency and violations detection must be carried on first, followed by resolving those inconsistencies and violations, since the resolution can only be done once the problems have been identified. Merging access control configurations and properties comes last, as it makes use of resolution strategies when a conflict arises during the merging process.

4.1 Incremental inconsistency and property violations detection

Incremental verification works well in general, but the specific case of access control, or even security in general, results in current approaches still selecting rules that do not need to be re-checked, especially when these rules happen to be quite complex and take a relatively long time to verify. If Egyed's approach [4] works very well for general consistency checking, there are cases where it will still select too many rules to be re-verified. It is usually not a problem, as consistency rules are generally relatively simple since they typically involve very few, often only two, model elements.Putting too much effort in detecting the rules that have to be re-verified may then actually take longer than verifying a few rules that did not need to be re-verified. Access control properties verification rules, on the other hand, usually involve more elements: if one wants to find out whether a user can perform a specific action, then it is necessary to at least go through all her/his roles and all the associated permissions. Depending on the method used for the verification, even more elements may be involved. It is also easy to detect some cases where, although a modified element is accessed by a rule during its verification, there is no need to re-evaluate the rule if it was previously verified. Let's take the first example that we mentioned in the introduction: a professor can edit his students' marks, but a student can only read his own, and can not edit any. A rule that determines whether a user can or can not edit a mark will have to check whether s/he has the roles allowing her/him to perform said action. If that property is verified for user u in version N of the software, then giving this user an additional role in version $N+1$ will only give her/him *more* permissions. That makes the re-

verification of the rule useless, even though the added role falls within the set of elements that are accessed by the rule.

Nentwich's xlinkit [19] also has several limitations, as it can only perform pairwise consistency checking, making it impossible to detect some inconsistencies.

An incremental verification approach targeted specifically to access control rules should take those cases into account and make sure that such rules are not re-checked, while still ensuring completeness, as all the rules that have to be checked are checked, to detect all the inconsistencies.

4.2 Inconsistency and property violations resolution

The suggestion of repair actions for access control properties brings two new challenges: first, to extend the completeness of the suggested repair actions, and second, to make sure that these actions do not lower the security level, unless this is really what the user wants to do.

The existing approaches, while all correct, are only complete within restrictive boundaries: the repair actions introducing new elements as well as those made of several atomic changes are not considered. For access control in particular, both these types of repair actions are important. Introducing new elements may be a very good way of making a model fulfil its access control properties, for example by activating a new role for a user before performing an action that requires several permissions. If we go back to our student marks system, we may want to make sure that a user with the role `Student` cannot edit his own marks, or even the marks of his fellow students. If the model allows the student to perform such an operation, it is therefore invalid. If we can add new elements to the model, then we can make sure that a new permission, that the `Student` role does not have, is necessary to perform a change in the marks. This will lead to a valid model, without compromising the security level. If, however, it is not possible to add new elements to the model, then the only option left is to lower the requirements and allow a student to change marks, leading to a model that, while valid, has a lower security level than originally intended.

Similarly, changes involving several atomic changes should be considered too, for example when a role needs to be separated in two different roles, each with its own set of permissions, and the users assigned to the original role reassigned to one or another of the resulting roles. Instead of only having students and professors, for example, one may want to introduce teaching assistants (TAs). TAs would be students who would be able to edit other student's marks, but marks edited by a TA would have to be confirmed by a professor. It would therefore be necessary to add a new role (one atomic operation), give it the appropriate permissions (N atomic operations, with $N \geq 1$), and assign the role to the users that should have it (M atomic operations, with $M \geq 1$). At least 3 operations would therefore be necessary. Since existing approaches will only suggest atomic changes instead of sequences, they will not be able to suggest this solution, although it is a valid one.

With multiple changes and new elements added comes another problem, though: completeness can not be guaranteed, as there is an infinite number of sequences of atomic changes that can be generated. A solution to this problem would be to allow one to limit the size of a sequence of changes. The bigger the maximum sequence, the more potential solutions can be found, but the longer the generation of those solutions will take. Completeness would still be limited, but the boundaries would be larger than with the existing solutions.

Ordering of the suggestions is another problem that will greatly benefit from being handled at a more specific level, since in access control, the smallest repair action is not necessarily the most desirable one. For example, in an invalid model that does not allow a specific user to perform a specific action, the smallest repair action may be to simply remove the access control requirement for said action. It would lead to a valid, but less secure, model. Other factors need to be taken into account, such as how many security properties will be impacted, whether the change will lead to higher and/or lower permissions for some classes of users, etc. These are by definition specific to access control.

4.3 Merging RBAC configurations and properties

Merging RBAC configurations and properties is another case where general approaches do not apply well to the specific problem of access control. First, it differs from the problem that most general approaches try to solve as the models to be merged do not share a common ancestor, making two-way merging the only available solution. In the second example mentioned in the introduction, two companies are merging. Since they are completely different companies, created at different times by different people, their access control configuration and properties have been developed independently from the very beginning. Second, the specificities of access control are not captured by the general approaches: in the example, each organisation has its own set of users, roles, permissions, and each one has different expectations about what a user should or should not be able to do. Merging role hierarchies, for example, can be very complicated. While identifying exact matches between roles in the organisations is relatively easy, identifying similar, yet not exactly identical roles is a much more complicated problem. For example, both companies may have a role `bank clerk`, but a completely different understanding of what a bank clerk can do: s/he could grant credits for up to £1000 without supervision, and up to £20000 as long as her/his direct supervisor approves the transaction, according to the rules set up by the first organisation, but would be able to grant a credit of up to £50000 without supervision according to the second company's rules. Even though both roles have the same name, they are quite different. Another problem may arise when two roles are different, say `bank clerk` in the first company and `financial adviser` in the second one, but it turns out that they have *almost* the same set of permissions. These may be merged together, but the decision of whether to merge them and how is down to the user. Unlike merging similar classes in a standard class diagram, where taking the union of the methods, attributes and associations of the two original classes in order to produce the merged class may be an acceptable strategy, it is not adapted to merging similar roles, as taking the union of their permissions would result in a merged role that may have more permissions than desired, therefore lowering the security level of the entire model. Instead, user input is necessary to determine which permissions are to be kept, and which are to be abandoned. Furthermore, this process may have an impact on the model verification, so it is also necessary to detect which rules would need to be re-verified.

Finally, in the case where both the functional model and the access control configuration and properties have to be merged, it is possible that merging the access control configuration first, and then only the functional model, may actually help for the merging of the functional model, as the access control configuration would provide additional information of which elements are the same and which elements are different.

5. CONCLUSION

Correctly specifying access control and evolving such specifications is an important problem for many organisations. We have reviewed existing work (including our own) and found that on the one hand Model-Driven Security Engineering approaches for access control provide little support for evolution, and on the other hand general model verification and evolution do not apply very well to the specific problem of access control. In particular, our analysis revealed the following shortcomings in existing approaches.

To detect access control inconsistencies and violations, users need

Smart rule checking: Access control rules, typically complex, should not be unnecessarily re-checked when the model changes, but naive heuristics are insufficient.

To resolve the detected inconsistencies and violations, users need

Security-aware correctness: Obtaining a consistent model is not enough, it must also fulfil the access control properties expressed.

Security-focused ordering: The possible resolution strategies should be ordered according to their impact on the security of the model.

Security impact visualisation: The designer must be able to immediately see and understand the consequences of a resolution strategy on the security of the model.

Bounded completeness: Some access control inconsistencies and violations can only be resolved by a transaction of atomic changes, but the transaction must be bounded to achieve completeness of the resolution strategy.

Flexible strategies: Suggested resolution strategies could also involve creating new model elements, to provide valid security hardening changes that would otherwise not have been considered.

To merge models and access control configurations and properties, users need

Role and permission similarities detection: The user should be pointed to roles and permissions that are very similar, making them good candidates for being merged, while still being able to see what the consequences of such a merge would be on the security of the model.

We therefore put forward the position that the above problems must be addressed by the research community in order to improve the support for evolving access control policies specified on system models.

6. REFERENCES

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

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security for process-oriented systems. In *Proc. symposium on Access control models and technologies*, pages 100–109. ACM, 2003.

[3] X. Blanc, A. Mougenot, I. Mounier, and T. Mens. Incremental detection of model inconsistencies based on model operations. In *Proc. Int'l conf. on Advanced Information Systems Engineering*, pages 32–46. Springer-Verlag, 2009.

[4] A. Egyed. Instant consistency checking for the UML. In *Proc. Int'l conf. on Software engineering*, pages 381–390. ACM, 2006.

[5] A. Egyed. Fixing Inconsistencies in UML Design Models. In *Proc. Int'l conf. on Software Engineering*, pages 292–301. IEEE, 2007.

[6] A. Egyed, E. Letier, and A. Finkelstein. Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In *Proc. Int'l conf. on Automated Software Engineering*, pages 99 –108. ACM, 2008.

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

[8] S. Judson, R. France, and D. Carver. Supporting rigorous evolution of uml models. In *Proc. Int'l conf. on Engineering Complex Computer Systems*, pages 128 – 137. IEEE, 2004.

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

[10] J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt. Incremental security verification for evolving umlsec models. In *Proc. European Conf. on Modelling Foundations and Applications*, pages 52–68. Springer, 2011.

[11] A. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Organization based access control. In *Proc. int'l workshop on Policies for Distributed Systems and Networks*, pages 120 – 131. IEEE, 2003.

[12] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proc. Int'l conf. on The Unified Modeling Language*, pages 426–441. Springer-Verlag, 2002.

[13] T. Mens. A state-of-the-art survey on software merging. *Software Engineering, IEEE Transactions on*, 28(5):449 –462, may 2002.

[14] T. Mens and T. D'Hondt. Automating Support for Software Evolution in UML. *Automated Software Engg.*, 7(1):39–59, 2000.

[15] T. Mens and R. V. D. Straeten. Incremental resolution of model inconsistencies. In *Proc. Int'l conf. on Recent trends in algebraic development techniques*, pages 111–126. Springer-Verlag, 2007.

[16] T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.*, 152:125–142, 2006.

[17] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In *Proc. Int'l Workshop on Principles of Software Evolution*, pages 13–22. IEEE, 2005.

[18] L. Montrieux, M. Wermelinger, and Y. Yu. Tool Support for UML-Based Specification and Verification of Role-Based Access Control Properties. In *Proc. joint meeting of the European Software Engineering Conference and ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2011.

[19] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Internet Technol.*, 2:151–185, May 2002.

[20] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta. Privacy-aware role-based access control. *ACM Trans. Inf. Syst. Secur.*, 13(3):1–31, 2010.

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

[22] R. Van Der Straeten, T. Mens, and S. Van Baelen. Challenges in model-driven software engineering. In *Models in Software Engineering*, volume 5421 of *LNCS*, pages 35–47. Springer, 2009.