Language-specific vs. language-independent approaches: embedding semantics on a metamodel for testing and verifying access control policies

Yves Le Traon, Tejeddine Mouelhi, Franck Fleurey, Benoit Baudry

Language-specific vs. language-independent approaches: embedding semantics on a metamodel for testing and verifying access control policies

Yves Le Traon
University of Luxembourg
Luxembourg

Tejeddine Mouelhi
Institut TELECOM ; TELECOM Bretagne ; RSM,
Université européenne de Bretagne, France

Franck Fleurey
SINTEF,
Oslo, Norway

Benoit Baudry
IRISA.INRIA
Rennes, France

***Abstract*— in this paper, we study an issue related to the abstraction level of a meta-model through the example of a model-driven approach for specifying, deploying and testing security policies in Java applications. The issue we focus on is the balance between a 'generic' meta-model and the semantics we want to attach to it, which has to be precise enough. The goal of the original work was to present a full MDE process to check the consistency of a security policy and generate qualification criteria for the test cases testing the security mechanisms in the final code. The most original idea is that security policy is specified independently of the underlying access control language (OrBAC, RBAC, DAC or MAC). It is based on a generic security meta-model which can be used for early consistency checks in the security policy. We qualify the test cases that validate the security policy in the application with a fault injection technique, mutation applied to access control policies. In the empirical results on 3 case studies, we explore the advantages and limitations of the mutation operators and verification checks whose semantics is defined on the meta-model. The overall question we address is not the feasibility of the approach as shown in our previous work but the quality of a metamodel for test and verification purpose.**

Keywords-Metamodeling, Security, MDE methodology

I. INTRODUCTION

An important issue in model-driven engineering has been recently tackled in [1], which concerns the intent of modeling. It focuses on the very heart of modeling, on the nature of relations, or on the patterns of relations that are discovered between these modeled 'things'. The authors define a canonical set of relations that can be used to ease and structure reasoning about modeling.

This workshop paper is related to this analysis, on the specific case of security test case generation and security verifications for access control policies. The intent of all model-based testing technique is to offer both:

- Models of high level of abstraction in order to simplify test reuse with the idea that models are easier to reuse, because they are platform independent,
- Test concretization mechanisms since, at the very end, the "abstract" test objectives have to be executed on a real specific platform and implementation.

In other words, the model-based testing domain must keep in mind the need for abstraction as well as the detail of implementation-specific features. This becomes even more difficult when we want to define a model-driven approach, where we have one more level of abstraction (instance of models, models instance of a metamodel, metamodel).

This paper aims at opening the discussion through the example we met in a previous research work [2]. More specifically, we would like to illustrate the difficulty of building a metamodel of the right level of abstraction for making several access control policy languages interoperable, and for applying them the same testing criteria and verification checks. We question the quality of the metamodel we propose for security test qualification and verification. If it is too abstract, the metamodel allows describing more models but will be able to carry less semantics. If it is too low-level, it will be close to a specific language and will be able to carry more of the semantics of this language but will not be applicable for other purposes. This problem is not new but we believe that this example highlights some interesting points that arise in the case of testing and verifying access control policies. We show that, for a meta-model which allows describing any of the possible rule-based access control languages, there are advantages and drawbacks which depend on its level of abstraction. The idea here is that we want to produce a metamodel which embeds the semantics of the test qualification criteria and the verification checks for the access control policies, whatever the specific access control formalism is.

Parts of the papers are a summary of the overall approach presented in [2], but the contribution of this paper is to highlight the advantages and drawbacks of using qualification criteria independent from the specific access control languages.

Section 2 recalls the approach and the metamodel for security validation and verification. Section 3 explains how we exploit this metamodel for language independent verification checks and security mutants generation. Mutation operator semantics is defined on the metamodel: the fault model is thus shared by any access control language, instance of the core metamodel. By construction, the validation is based on a common certification basis. Finally, Section 4 presents the empirical results on three Java case studies and pinpoints some advantages and limits of such a semantic enriched metamodel, compared to language specific techniques.

II. METAMODELLING FOR SECURITY VALIDATION AND VERIFICATION

We recall the generic verification and validation techniques we propose in [2] that are independent of any particular security formalism.

A. *Principles of the MDE for security approach*

To "embed" semantics elements in the metamodel, we use the Kermeta language which has been designed for that purpose [3]. Since several access-control formalisms/languages have been developed over the past decades, including RBAC [4], OrBAC [5], DAC [6] or MAC [7, 8]), we reuse the metamodel presented in [2], which allows expressing all of these rule-based access control languages. In essence, these languages provide means to describe under the form of access control rules, the subjects' permissions and prohibitions to access a resource (for instance, the right to configure a firewall or to access a specific service or record in a database).

The proposed MDE process is based on a domain-specific language (DSL) to model security formalisms/languages as well as security policies defined according to these formalisms. This DSL is based on a generic metamodel that captures all the necessary concepts for representing rule-based access control policies. The MDE process relies on several automatic steps in order to avoid errors that can occur with repeated manual tasks. We do not detail this process in this paper since it has been published in detail in [2]. This includes the automatic generation of a specific security framework, the automatic generation of an executable PDP (Policy Decision Point) from a security policy and the injection of PEPs (Policy Enforcement Point) into the business logic through aspect-oriented programming. The Policy Enforcement Point is the point in the business logic where the policy decisions are enforced. It is a security mechanism, which has been intentionally inserted in the business logic code. On call of a service that is regulated by the security policy, the PEP sends a request to the PDP to get the suitable response for the requested service by the user and in the current context. The Policy Decision Point encapsulates the Access Control Policy and implements a mechanism to process requests coming from the PEP and return a response which can be deny (discard access) or permit (grant access).

Since this interaction of the PDP with the business logic can be faulty (e.g. a hidden security mechanism may bypass some PEPs [9]), a qualification environment is provided which performs (1) a priori verifications of security models before PDP component and PEP generation, (2) a posteriori validation of the test cases for the implementation of the policy. This qualification environment is independent of security policy languages and is based on a metamodel for the definition of access control policies. It provides model transformations that make the qualification techniques applicable to several security modeling languages (e.g. RBAC, OrBAC). The interest of metamodelling V&V artifacts (structure and semantics) is that the same certification process is applied for testing the system even when it combines policies expressed in different languages.

We also focus on the modeling levels involved in this full-MDE process, and especially on the original idea to express semantics (fault models, mutation operators and verification functions) at meta-level independently from the chosen access control language. We will discuss the limitations of a too abstract metamodel, which do not allow embedding and performing advanced security checks.

B. *Overview of the modeling architecture*

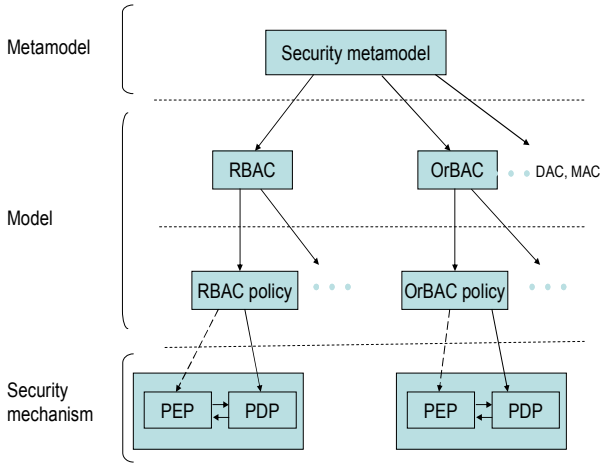

Figure 1. The modeling framework

Figure 1 depicts the different levels of models we manipulate in this paper. The security metamodel captures common concepts that are needed to define security formalisms (such as RBAC, OrBAC access control languages). The core security metamodel can be instantiated to define a security policy formalism and the expression of rules in this formalism.

The security model is a platform independent model which captures the policies defined in the requirements of the system. In practice, the meta-model allows the type of rules to be modeled as well as the rules. In [2], we detail all the steps, from access control policy definition to the generation of PDP and PEPs. A careful validation of the resulting code with respect to the security model is required. We define the structure and the semantics of V&V artifacts that ensure the consistency and the correctness of the security mechanisms.

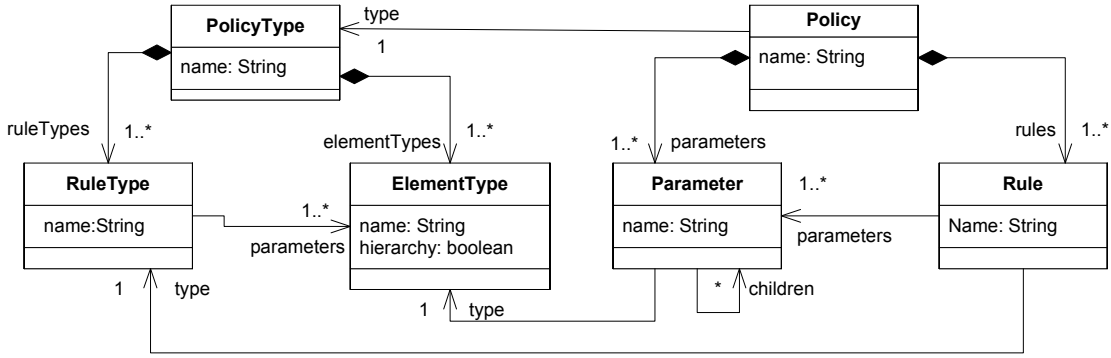

Figure 2. The meta-model for rule-based security formalisms

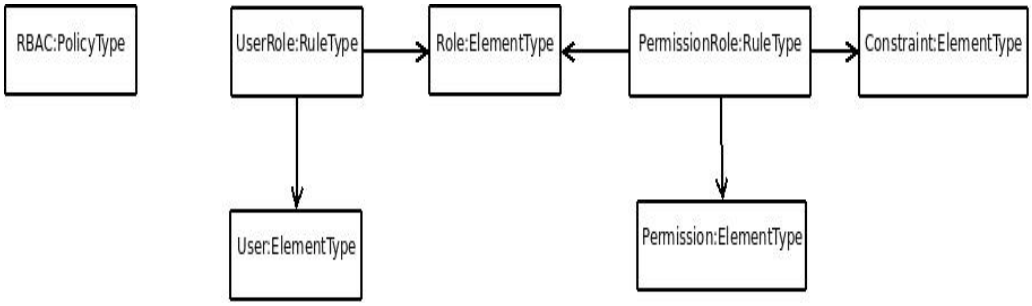

Figure 3. RBAC model

C. *The access control metamodel*

In the literature, several access control formalisms such as RBAC, OrBAC, etc. are based on the definition of security rules. All these formalisms allow the definition of rules that control the access to data, resources or any type of entity that should be protected. The formalisms differ by the type of rules and entities they manipulate. In order to provide generic validation mechanisms, this work is based on a generic access-control metamodel which allow both capturing the specificities of a particular access-control formalism and corresponding access-control policies. This section introduces this generic access-control metamodel and illustrates how it supports the definition of RBAC policies.

Figure 2 recalls the meta-model we study in this paper. The meta-model is divided into two parts, which correspond to two levels of instantiation:

The first level of instantiation (classes POLICYTYPE, ELEMENTTYPE and RULETYPE) allows modelling particular access-control mechanisms such as (RBAC OrBAC, MAC or DAC). A POLICYTYPE defines a set of element types (ELEMENTTYPE) and a set of rule types (RULETYPE). Each rule type has a set of parameters that are typed by element types.

The second level (classes POLICY, RULE and PARAMETER) allows instantiating a specific security policy using a defined formalism. A POLICY must have a type (an instance of POLICYTYPE) and defines rules and parameters. The type of a policy constrains the types of parameters and rules it can contain. Each parameter has a type which must belong to the element types of the policy type. If the *hierarchy* property of the parameter type is true, then the parameter can contain children of the same type as itself. Policy rules can be defined by instantiating the RULE class. Each rule has a type that belongs to the policy type and a set of parameters whose types must match the types of the parameters of the type of the rule.

In practice, the two parts of the metamodel have to be instantiated sequentially: first define the formalism, and then define a policy according to that formalism.

D. *Instantiating the metamodel*

The fact that the proposed meta-model supports both modelling security formalisms and security policies in these formalisms makes it possible to represent any rule-based security policy. Any rule-based formalism has to be modelled using the part of the meta-model which captures policy types in order to be supported. This has to be done once for any security formalism in order to be able to represent corresponding security policies with the policy part of the proposed meta-model. This section demonstrates RBAC have been modelled using the proposed approach.

Figure 3 presents the RBAC model as expressed using our metamodel. In this work, we consider an RBAC model augmented with constraints over permissions. It defines four types of entities: users, permissions, roles and constraints. The model associates users with roles on one hand and roles with permissions on the other hand. Two types of rules have to be defined:

UserRole rules which have two parameters: a user and a role.

RolePermission rules which have three parameters: a role, a permission and a constraint.

A simple access control policy was modeled based on RBAC. It includes:

- Three users: alice, yves and romain.
- Three roles: Student, Director and Secretary.
- Three permissions: BorrowBook, ModifyAccnt and CreateUserAccount.
- Two constraints: WorkingDays and Holidays.

Six rules were defined to associate users with roles on the one hand and associate permissions with roles on the other hand:

POLICY LibraryRBAC (RBAC)

R1 -> UserRole(romain Student)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R5 -> RolePermission(Personnel ModifyAccnt WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

III. V&V DERIVATION FROM THE SECURITY METAMODEL

The verification of the consistency of a given security policy is performed at early stage, during the specification by security experts, who express who can access to what. The validation process then aims at qualifying the test cases used to ensure the correctness of the security mechanisms. The fault model is defined at the meta-level and can be executed to inject faults into security policies.

A. The verification checks

Verifications are performed in order to check the soundness of the security policy and its adequacy with regards to the requirements of the application. In particular, we detect conflicts between rules, which is a tedious task. A simple case of conflict occurs when a specific permission is granted by a rule and denied by another. These verifications also include checking that each business operation can be performed by at least one type of user or that a specific set of operations can only be performed by specific users (e.g. administrators). All these verifications are carried out by the security metamodel and are thus language independent: verifications and transformations are generic and apply to all types of rule. The main verification is offered by construction, when the conformity of a security model to its metamodel is checked. In this way, we have a minimum consistency that is obtained with the conformance relationship of a model to its metamodel. We also add some extensible verification functions. They constitute preconditions which are checked before deploying the policy and generating the mutants. Three verification functions are implemented:

- `policy_is_conform()`: the policy conformance to the underlying policy type (OrBAC or RBAC etc.). In order to guarantee that the defined rules meet the types of parameters of rules defined by the policy type. The flexibility of the model can lead to having incorrect rules which do not conform to the types of rules supported by the policy.
- no_conflicts(): checks the absence of conflicts. It essentially involves checking that there are no rules having the same parameters and having different types. This is especially useful for OrBAC where both prohibition and permission rules can be defined.
- no_redundancies(): checks that the security policy is minimal, which means that no rule appears more than once. This inconsistency can happen when high level hierarchy parameters are used to specify rules in addition to the others descendant parameters, leading to have two or more of the same rule for that descendant parameter.

The conformance verification function detects simple erroneous rules such as wrong parameters or wrong numbers of parameters.

This verification step is important in order to detect faults in the specified policies. These faults are detected early during the modeling and are corrected before the deployment and the generation of the XACML policy. XACML is an OASIS standard [10] for expressing policy using the XML language.

B. Security test qualification by mutation testing

To properly validate the security of the application, test cases have to cover all the security features of the application. Mutation testing is a test qualification technique introduced in [11] which has recently been adapted for security testing [12, 13]. The intuition behind mutation testing applied to security is that the security tests are qualified if they are able to detect any elementary modification in the security policy (mutants). From the initial policy a set mutant policies is generated using the mutation operators. Then, test cases are executed against these mutants in order to check if they are able to find the injected faults. A mutant is said to be killed if at least one test case detects the presence of the seeded fault.

The originality of the proposed approach is to perform mutations on the platform independent security model using generic mutation operators. If the tests are not able to catch a mutant then new test cases should be added to exercise the part of the security policy which has been modified to create this mutant. In practice the undetected mutants provide valuable information for creating new tests and covering all the security policies.

To have a common certification process, we generate mutants from mutation operators defined on the generic metamodel. The idea is to extend the security metamodel with the definition of mutation operator, using Kermeta [3]. Kermeta is an open source metamodelling environment developed by the Triskell team at IRISA that is fully integrated

with Eclipse. It has been designed as an extension to the meta-data language EMOF [14] with an action language that allows specifying semantics and behavior of metamodels. An instance of such a metamodel automatically embeds this semantics and behavior: this is this facility offered by Kermeta we exploit in this paper to define the mutation operators' semantics at meta-level.

The security policy languages (such as RBAC on the figure, or any other access control language) are instances of the metamodel and thus embed the way a specific security policy can be mutated. Code generation from the mutants security policies create as many faulty PDPs as there are mutant security policies. The test qualification process can then be applied on the set of faulty systems, embedding mutant PDPs.

We define five mutation operators for security policy testing. These operators are defined only in terms of the concepts present in the security metamodel, which means that they are independent of a specific security formalism. Thus, these operators can be applied to inject faults into any policy expressed with any formalism defined as an instance of our metamodel. The definition of mutation operators at this meta-level is critical for us since it allows the qualification of test cases with the same standard, whatever the formalism used to define the policy. In order to generate faulty policies according to these operators, we have added one class to the metamodel for each operator.

- **RTT:** Finds a first rule type that has the same parameter as the type of another rule type. Then it replaces the rule parameter of one rule having the first rule type with the other rule type.
- **PPR:** Chooses one rule from the set of rules, and then replaces one parameter with a different parameter. It uses the knowledge provided by the metamodel (by ruleType and parameterType classes) about how rules are constructed.
- **ANR:** Uses the knowledge about the defined parameters and the way rules are built. Then it adds a new rule that is not specified.
- **RER:** Chooses one rule and removes it.
- **PPD:** Chooses one rule that contains a parameter that has descendant parameters (based on the parameter hierarchies that are defined) then replaces it with one of the descendants. The consequence here is that the derived rules will be deleted and only the rule with the descendant parameter remains.

The Kermeta action language is imperative and object-oriented and is used to provide an implementation of operations defined in metamodels. It includes both OO features and model specific features. Convenient constructions of the Object Constraint Language (OCL) such as closures (e.g. each, collect, select) are also available in Kermeta. Figure 4 shows the operator classes. The `mutate()` method is implemented in [3]. Figure 5 shows the implementation for the RER operator.

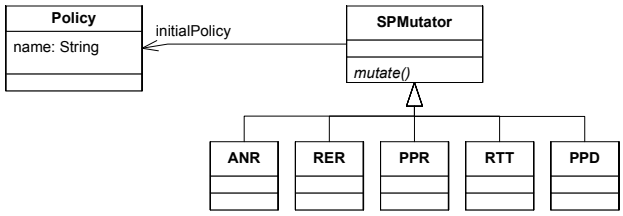

Figure 4. Extension of the security metamodel with mutation operators

The `mutate()` method is implemented in Kermeta. What is important to notice in this method is that it is defined only using concepts defined in the metamodel. Thus, this method can generate a set of mutated policies, completely independently of the formalism they are defined with.

```
class RER inherits SPMutator {


    method mutate(p : Policy) : set Policy[*] is do
        var mutant : Policy
        result := Set<Policy>.new
        // loop on rules
        p.rules.each{ r |
            // create mutated policy
            mutant := p.copy
            mutant.name := p.name + "-RER-" + r.name
            // remove one rule
            mutant.rules.remove(mutant.rules.detect{x | x.name == r.name})
            // adds the mutant policy
            result.add(mutant)
        }
    end
}
```

Figure 5. The RER operator

It is important to notice that the impact of the mutation operator depends on the access control formalism used to define a policy. The faults that are simulated are very different as shown in the examples. The same operators emulate very different flaws in the policies. For instance, the ANR operator applied to RBAC simulates the addition of a new permission, while OrBAC will simulate the addition of a new prohibition or a new permission. The impact of the operator depends on the semantic and the logic of the access control formalism.

C. *Generated mutants*

In order to illustrate the mutants we get when we apply our approach, we show examples of RBAC mutants.

1) *RBAC mutants*

Next, we present some examples of mutants related to an RBAC policy. The initial RBAC policy is presented bellow:

POLICY LibraryRBAC (RBAC)

R1 -> UserRole(romain Student)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R5 -> RolePermission(Personnel ModifyAccnt WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

Here are some examples of the generated mutants

RER mutant:

POLICY LibraryRBAC-RER-R5 (RBAC)

R1 -> UserRole(romain Student)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

PPR mutant:

POLICY LibraryRBAC-RDD-R1-Student-Personnel (RBAC)

R1 -> UserRole(romain Personnel)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R5 -> RolePermission(Personnel ModifyAccnt WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

IV. DISCUSSION ABOUT LANGUAGE-INDEPENDENT V&V VS. LANGUAGE SPECIFIC ONES

In [2], we presented the applicability of our approach and provide test results and mutation scores for three systems :

- LMS: A Library Management System.
- VMS: A Virtual Meeting System.
- ASMS: An Auction Sale Management System.

Table I shows the size of the 3 applications (the number of classes, methods and lines of code LOC). Here we focus on the limitations and drawbacks of the use of a generic metamodel for expressing rule-based access control languages. The issue we met was the following:

- Sharing V&V treatments in a language-independent model permits common certification to be performed
- Sharing these V&V does not allow dealing with complex and specific potential flaws.
- The result is that a trade off must be found between genericity and specificity in case of intensive use of MDE.

TABLE I. COMPLEXITY OF THE THREE CASE STUDIES

	# classes	# methods	LOC (executable statements)
LMS	62	335	3204
VMS	134	581	6077
ASMS	122	797	10703

A. *Limitations of the verification*

The verification steps we propose allow detecting simple inconsistencies but cannot replace language-specific verifications, such as the ones performed for example for access control models like Or-BAC with a tool called MotOrBAC [15]. Access control languages do not usually provide such specific verification environment which makes our approach useful. Thus, the verification functions we propose have the advantage of being embedded in the metamodel. For instance, compared to a direct XACML code production, generic verifications offer a systematic way of detecting inconsistencies. In Table II, we highlight the relevance of the proposed verification functions for each of the targeted access control policy language. We notice that the no-conflicts verification is only useful for OrBAC policies, even if the semantics of this verification function is attached to the metamodel and is thus generic. If a new access control language including the notions of prohibition was modeled using the security metamodel, it would embed this useful verification function. Concerning the no-redundancies verification, while it has a meaning for each language even if the relevance is weak when the language includes no hierarchy (it only detects that there is no identical rules in the security policy). In conclusion, we have the following paradoxes:

- Some verification checks expressed on the meta-model have no meaning for a specific language
- Most advanced checks (e.g. logic conflicts detection, behavioral properties) cannot be expressed on our meta-model due to the fact the specific concepts of access control languages are not captured by the meta-model,
- The mutation operators we propose may be used to create faults for any language which is based on rules, even if it has no link with security and access control. In fact, if we analyse the metamodel we propose, we notice that there is no notion specific to security, but only concepts to capture the notion of rule and its parameters.

The solution to that problem might be metamodels composition/specialization. It would allow attaching partial semantics elements to each metamodel. For example, if we manipulate a real-time language with access control operations, we would like to combine in the same metamodel real-time notions and access control notions, as well as their semantics. Each partial metamodel could focus on each aspect and the final metamodel would be obtained by composition of these elements.

TABLE II. RELEVANCE OF GENERIC VERIFICATION FOR SPECIFIC ACCESS CONTROL LANGUAGES

	Policy_ is_conform	No_ conflicts	No_ redundancies
RBAC	y	n	y
OrBAC	y	n	n
OrBAC	y	y	y
DAC	y	n	n
MAC	y	n	n

1) Language-independent vs. language specific mutation-based qualification

Table III compares the number of mutants we obtain with OrBAC policies using a specific approach instead of the generic approach (based on the language-independent metamodel presented in [2]).

TABLE III. ORBAC SPECIFIC MUTANTS VS. GENERIC MUTANTS

System	generic mutants	specific mutants
LMS	1044	371
VMS	1572	1426
ASMS	3088	2056

Table IV is more important since it presents the mutation scores, with functional and security test cases, obtained with the generic and the specific approach. The specific approach has been presented in [12, 16] and it benefits the MotOrBAC tool used to generate mutants.

TABLE IV. ORBAC MUTATION RESULTS VS. GENERIC MUTATION RESULTS

Mutants	Basic Mutants (func. Tests)		
System	LMS	VMS	ASMS
Generic mutants	72%	61%	45%
Specific mutants	78%	69%	55%
Delta	-6%	-8%	-10%

Mutants	ANR mutants (sec. tests)		
System	LMS	VMS	ASMS
Generic mutants	13%	12%	28%
Specific mutants	17%	19%	33%
Delta	-4%	-7%	-4%

The delta reveals that, for all cases, the variation of mutation scores is lower than 10%. This delta is due to the generation of more mutants with the generic approach, and reflects the proportion of equivalent mutants when using the generic approach. We could discuss if the lack of quality of some mutants generated using the MDE process is counter-balanced by the interest of having a certification process that is language-independent. Moreover, this distance that separates generic and specific can be reduced by the addition of a mutant filtering function at the language level that will remove irrelevant mutants.

V. RELATED WORKS AND CONCLUSION

In this paper, we do not want to recall the related works we did about security testing in [2]. Concerning security modelling, several works exist (UMLsec [17], SecureUML [18]). In [19], France et al. propose composition mechanism to build a design embedding access control notions.

An advanced approach is proposed by UMLsec [17], which extends UML model security requirements in UML diagrams. More precisely the approach introduces a UML profile allowing completing UML diagrams (such as activity diagrams and statecharts). The main benefit of this approach resides in providing formal methods and techniques for a thorough analysis of security in UML diagrams. The security requirements that are included in the models help evaluating these models and finding possible flaws in the design, while in our approach the model is used to build testing artefacts (policy mutants) that then will be used to validate the implementation.

Lodderstedt et al. proposed SecureUML [18] which is close to our contribution, especially concerning the generation of security components from dedicated models. The approach proposes a security modeling language to define the access control model. The resulting security model is combined with the UML business model in order to automatically produce the access control infrastructure. More precisely, they use the Meta-Object facility to create a new modeling language to define RBAC policies (extended to include constraints on rules). They apply their technique in different examples of distributed system architectures including Enterprise Java Beans and Microsoft Enterprise Services for .net. The two processes differ since we do not merge the security and the business models but generate security components independently focusing on different aspects. We consider the functional design and the deployment of the access control model as independent processes which are merged at the end. In addition, we consider the validation of the implementation and include it in our process. Moreover, our approach is generic and independent of the underlying access control model while SecureUML focus on RBAC model.

Our model-driven security approach has its advantages and its drawbacks. Our approach allows the access control model to be defined in a generic way, independently of the underlying rule-based access control language. We define a fault model at a generic level and use it to qualify the security tests needed to validate the implementation of the security policy. We also express verification checks at the meta-model level. In both

cases, we used Kermeta language to attach these elements of semantic to the meta-model. The feasibility of the approach and the benefits of common V&V approaches are both illustrated by applying the approach to 3 case studies. We presented and discussed the price to pay for genericity and multi-formalisms.

The objective of this paper is to discuss the problem of what is lost, in terms of specificity, when we go through a metamodeling process. This is a classical issue that is experienced by many people in model-driven engineering. This problem becomes critical when the models have to become concrete, and this is especially the case in the domain of software testing. To our knowledge, the first paper which opens the discussion about the intent of modeling is the paper of Muller et al. [1]. Some other works may exist on that fundamental subject, which is not our main research area. However, this paper focuses more on the relations between models than on the relationship/distance separating the representation (model, abstraction) from reality. We believe that this issue is a major problem in using MDE approaches, and that there are no predictable way/estimate to decide whether applying a model-driven approach (to manage a given test-generation or verification in a reusable way) will be fruitful or not. Predicting the quality of a metamodel would be highly desirable: it means been able to estimate its distance from the 'thing' it represents, to measure its adequacy with the role it has been built for.

Acknowledgments: This work is supported by "Région Bretagne" (Britanny Council) through a contribution to a student grant.

VI. REFERENCES

1. Pierre-Alain Muller, Frédéric Fondement, and Benoît Baudry, *Modeling Modeling*, in *Model Driven Engineering Languages and Systems*. 2009. p. 2-16.
2. T. Mouelhi, F. Fleurey, B. Baudry, and Y. Le Traon, *A model-based framework for security policy specification, deployment and testing*, in *MODELS 2008*. 2008.
3. Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. *Weaving executability into object-oriented meta-languages*. in *MoDELS'05*. 2005. Montego Bay, Jamaica: LNCS.
4. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, *Proposed NIST standard for role-based access control.* ACM Transactions on Information and System Security, 2001. **4**(3): p. 224–274.
5. A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin, *Organization Based Access Control*, in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. 2003.
6. B. Lampson. *Protection*. in *5th Princeton Symposium on Information Sciences and Systems,*. 1971.
7. K. J. Biba, *Integrity consideration for secure computer systems*, in *Tech. Rep. MTR-3153, The MITRE Corporation,*. 1975.
8. D. E. Bell and L. J. LaPadula, *Secure computer systems: Unified exposition and multics interpretation*, in *Tech. Rep. ESD-TR-73-306, The MITRE Corporation*. 1976.
9. Y. Le Traon, T. Mouelhi, A. Pretschner, and B. Baudry, *Test-Driven Assessment of Access Control in Legacy Applications*, in *ICST 2008: First IEEE International Conference on Software, Testing, Verification and Validation*. 2008.
10. *XACML:* http://www.oasis-open.org/committees/xacml/. [cited.
11. R. DeMillo, R. Lipton, and F. Sayward, *Hints on Test Data Selection : Help For The Practicing Programmer.* IEEE Computer, 1978. **11**(4): p. 34 - 41.
12. T. Mouelhi, Y. Le Traon, and B. Baudry, *Mutation analysis for security tests qualification*, in *Mutation'07 : third workshop on mutation analysis in conjuction with TAIC-Part*. 2007.
13. E. Martin and T. Xie. *A Fault Model and Mutation Testing of Access Control Policies*. in *Proceedings of the 16th International Conference on World Wide Web*. 2007.
14. OMG. *MOF 2.0 Core Final Adopted Specification.* 2004 [cited 2005; Available from: http://www.omg.org/cgi-bin/doc?ptc/03-10-04.
15. *MotOrBAC:* http://motorbac.sourceforge.net/index.php?page=home&lang=en. [cited.
16. Y. Le Traon, T. Mouelhi, and B. Baudry, *Testing security policies : going beyond functional testing*, in *ISSRE'07 : The 18th IEEE International Symposium on Software Reliability Engineering*. 2007.
17. J. Jürjens. *UMLsec: Extending UML for Secure Systems Development*. in *Proceedings of the 5th International Conference on The Unified Modeling Language*. 2002.
18. Torsten Lodderstedt, David Basin, and Jürgen Doser. *SecureUML: A UML-Based Modeling Language for Model-Driven Security*. in *Proceedings of the 5th International Conference on The Unified Modeling Language*. 2002.
19. Indrakshi Ray, Robert France, Na Li, and Geri Georg, *An aspect-based approach to modeling access control concerns.* Information and Software Technology, 2004. **46**(9): p. 575-587.