Issues in Representing Domain-Specific Concerns in Model-Driven Engineering

Lionel Montrieux, Yijun Yu, Michel Wermelinger
Centre for Research in Computing
The Open University
Milton Keynes, UK
{Lionel.Montrieux,Yijun.Yu,Michel.Wermelinger}@open.ac.uk

Zhenjiang Hu
National Institute of Informatics
Tokyo, Japan
Hu@nii.co.jp

***Abstract*—The integration of domain-specific concepts in a model-driven engineering (MDE) approach raises a number of interesting research questions. There are two possibilities to represent these concepts. The first one focuses on models that contain domain-specific concepts only, i.e. domain-specific modelling languages (DSML). The second one advocates the integration of domain-specific concepts in general-purpose models, using what we will refer to in this paper as domain-specific modelling annotation languages (DSMAL). In this position paper, we argue that each approach is particularly suited for specific activities and specific actors, and show how they can be developed and used together. We also highlight the challenges created by the use of two representations, such as the evaluation of models OCL constraints and the synchronisation between the two representations. As an illustration, we present `rbacUML`, our approach for integrating role-based access control (RBAC) concepts into an MDE approach.**

***Index Terms*—MDE, RBAC, OCL, DSML, DSMAL, UML, profile, transformation**

I. Introduction

In the past twenty years, researchers have explored model-driven engineering (MDE), a paradigm which makes use of increasingly refined models until code is produced. UML [1] is arguably the most widespread general-purpose modelling language used in MDE. UML features an extension mechanism, UML profiles [2], that allow one to extend the semantics of some UML elements through annotations, called *stereotypes*, and constraints written in OCL [3].

Some researchers have also focused their efforts on representing and integrating domain-specific concepts in their MDE approaches. This can be approached in two ways. The first one advocates the creation of models that *exclusively* contain domain-specific concepts. Such languages are called domain-specific modelling languages (DSML), and they produce domain-specific models. The second way prefers to annotate general-purpose models by extending general-purpose modelling languages. We call such extensions domain-specific modelling annotation languages (DSMAL), and they produce general-purpose models *annotated with* domain-specific concepts.

In this paper, we argue that both approaches can be used together, and that the use of one representation over the other depends on both the activity being performed, and the actor performing it. We present `rbacUML`, our approach for modelling Role-based access control (RBAC) concepts on models, and show how, from the same domain model, we have derived two UML profiles: one for a DSML, and one for a DSMAL. We discuss the relationships between DSMLs and DSMALs, and make the case that bidirectional transformations between the two representations are needed. We also discuss OCL constraints: the translation of constraints between both representations, as well as the ideal representation to evaluate each single constraint. Furthermore, we describe how a single UML model could be annotated with several DSMALs, and what the implications are in terms of synchronisation and potential conflicts.

The remainder of this paper is organised as follows: Section II presents the background on MDE, DSMLs and RBAC. We then present our solution for RBAC modelling, with both the DSMAL and the DSML, in Section III. Section IV discusses the transformations between the two representations, whilst Section V discusses the place of OCL constraints. Section VI concludes the paper.

II. Background

A. Model-Driven Engineering

Model-driven engineering (MDE) is the software engineering approach that uses models to reason about software. Models are defined according to a metamodel, but the growing number of metamodels led to another higher level to describe metamodels: meta-metamodels [4]. Arguably the most widely used MDE framework is the OMG's Model-driven architecture approach [5], which includes UML (Unified Modeling Language) models [1], OCL (Object Constraint Language) constraints [3] and MOF (Meta-Object Facility) metamodels and meta-metamodels [6].

In the security world, Fernandez-Medina et al. [7] 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*", but it is a direction that is currently being developed, including by Basin et al. [8], 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*".

978-1-4673-6447-8/13 © 2013 IEEE

MiSE 2013, San Francisco, CA, USA

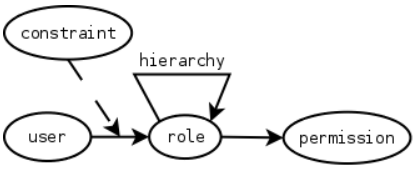

Fig. 1: RBAC model

B. Domain-Specific Modelling Languages

Domain-specific modelling languages (DSML) are the modelling equivalent of domain-specific languages (DSL) [9], i.e. they are modelling languages whose purpose is to efficiently represent concepts from a particular domain. In the literature, there are essentially two schools of thought when it comes to representing DSMLs: either using "ad-hoc" representations, or using UML profiles, which are extensions of the UML metamodel that add additional semantics to elements via stereotypes and tagged values. UML profiles can also embed OCL constraints that enforce some properties of the domain. Selic proposes an excellent method to implement a DSML using an UML profile [2], and many researchers have studied the relationships between "ad-hoc" DSMLs and those implemented using UML profiles. For example, Abouzahra et al. use model transformations and model weaving to bridge the two approaches [9], while Wimmer proposes a semi-automated approach to generate a UML profile given a DSML and a mapping from the DSML to the created profile [10]. Others, such as Giachetti et al., use a hybrid representation to allow for the interchange between "ad-hoc" DSMLs and UML profiles [11].

C. Role-Based Access Control

Traditional access control models [12] allow the administrator to assign permission directly to users. This makes the maintenance of large access control directories difficult. By contrast, role-based access control (RBAC) [13] forbids the direct assignment of permissions to users, and introduces the concept of *roles* between users and permissions. Roles in RBAC are meant to match actual roles in an organisation. Permissions are assigned to roles, which are assigned to users.

The RBAC standard also defines other constructs: role hierarchies, where a role inherits its ancestors' permissions, and static (resp. dynamic) separation of duty, where two roles cannot be assigned to (resp. simultaneously activated by) the same user. Fig. 1 illustrates the RBAC model.

III. The rbacUML Approach

`rbacUML` is our approach for integrating RBAC into existing MDE processes. It is implemented as a plugin [14] for IBM Rational Software Architect [15], a well-known UML modelling environment built on top of Eclipse. In this section we focus on the parts of `rbacUML` most relevant to this paper.

We started out by defining the `rbacUML` DSMAL, which we will present first after introducing the domain-specific metamodel. As we were expanding the features of the DSMAL, we realised that a DSML would be more suitable for some activities or actors.

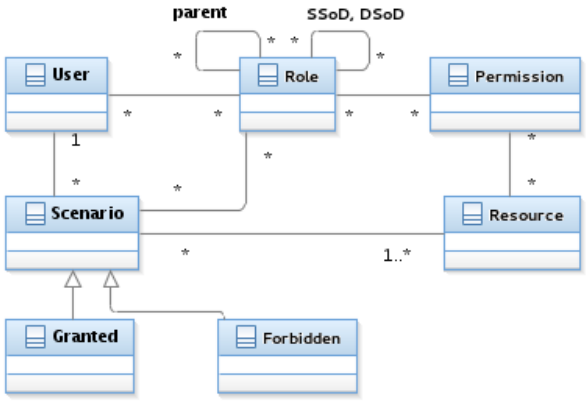

Fig. 2: `rbacUML` metamodel

A. The Metamodel

Both representations stem from the same metamodel, shown in Fig. 2. The top part represents the entire RBAC standard, with users, roles, permissions and their assignments; role hierarchies; static and dynamic separation of duty (SSoD and DSoD) constraints. On the bottom right we have the resources whose access needs to be restricted. To access a resource, one needs *all* of its associated permissions. On the bottom left are the scenarios. They are instances of requirements that the rest of the model needs to satisfy. There are two types of scenarios (represented on the figure as only one `Scenario` element): the first one requires that a user, given a set of activated roles, must be able to access a set of resources. The second one requires that a user, given a set of activated roles must *not* be able to access at least one of the resources in the set. From this metamodel, we have derived, using Selic's method [2], both the DSMAL and the DSML.

B. The DSMAL

Although Selic's method [2] focuses on the implementation of DSMLs with UML profiles, we found that it also applies to DSMALs. We followed it to implement the DSMAL for `rbacUML`, whose extension of the UML metamodel is shown in Fig. 3. For the DSMAL we want to annotate *existing* elements with RBAC-related concepts at *every* place where it is important for the designer to know that RBAC concepts have an influence. Hence, there is some degree of redundancy. Indeed, in a general purpose model in UML, it is likely that the designer will have represented several views, or diagrams, of the system to be built. RBAC concepts should appear on each of the relevant diagrams. `rbacUML` annotations can be divided in three categories. First is the *configuration*, which defines the users, roles (which match the organisation's structure) and permissions as well as their assignments and the SoD constraints. The elements are represented as classes and

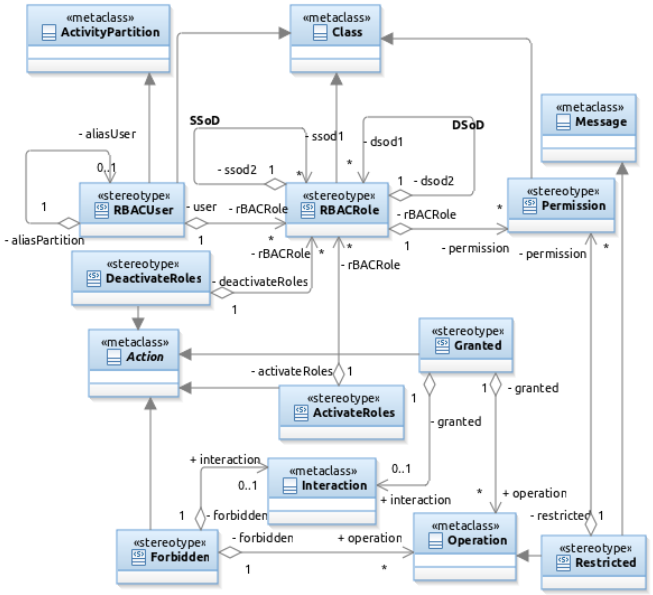

Fig. 3: Extension of the UML metamodel for `rbacUML`

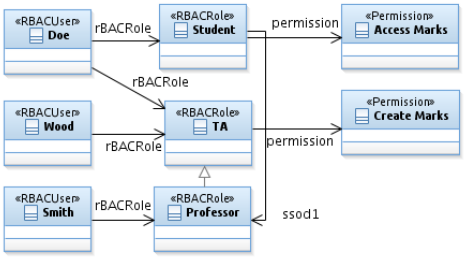

Fig. 4: Access Control diagram (configuration)

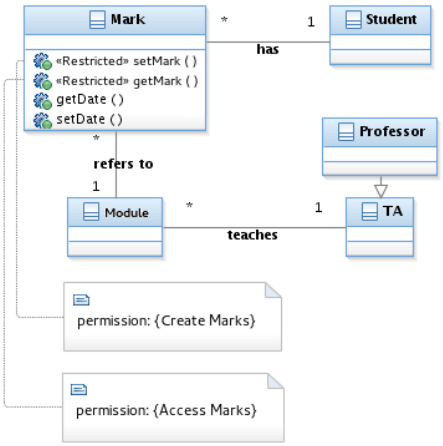

Fig. 5: Class diagram (policy)

the assignments and constraints as associations. Second, the *policy*, which defines the resources to be protected. Resources are represented by operations in class diagrams, and message calls in sequence diagrams. Third, the *scenarios*, represented by activity diagrams. The actual scenario is represented by an action, which sits in a partition that represents the user performing the action. Figs. 4, 5 and 6 are an example of a model built with the DSMAL. They represent a simple grading application for a university where professors and teaching assistants (TAs) can update their students' marks, and the students can consult their own marks. Fig. 4 is the configuration, and shows the users, roles, permissions and their assignments. There is also a role hierarchy relationship between `Professor` and `TA`, and a static separation of duty constraint between the `Student` and the `Professor` roles. Fig. 5 is the class diagram, where resource whose access must be restricted are represented by operations stereotyped with `Restricted`. With the stereotype come associations to each of the permissions that are required to execute the operation. This diagram represents the policy. Finally, the activity diagram in Fig. 6 represents two scenarios. Each scenario is represented by an action that sits in an activity partition that represents the user.

The separation between different views, as well as the redundant elements, is quite obvious in this model. Two users are represented twice: in the access control diagram, and in the activity diagram. There is also a case of one concept from the initial metamodel being *split* into several ones in the general purpose annotations: the roles activation for a scenario. The `rbacUML` DSMAL includes 34 OCL constraints.

The DSMAL is very well-suited for designers to integrate RBAC into a software model. By mixing RBAC-specific concepts with general-purpose concepts in UML, they can immediately see where authorisation constraints will affect the software being built. However, we quickly came to realise that such a level of detail isn't always welcome. A system administrator will typically maintain the RBAC configuration. He doesn't need to see the entire model, but only needs to have access to the RBAC-specific concepts. Furthermore, the DSMAL is complex, and when we started working on automated fixing of errors in the model, we realised that a model that only contains RBAC-specific concepts would be

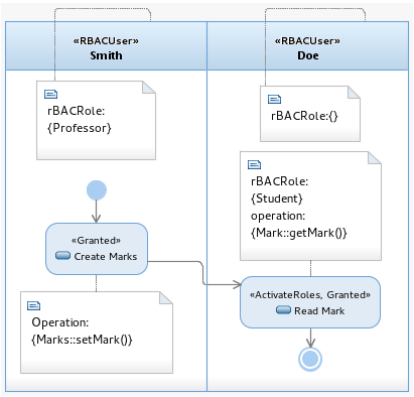

Fig. 6: Activity diagram (scenarios)

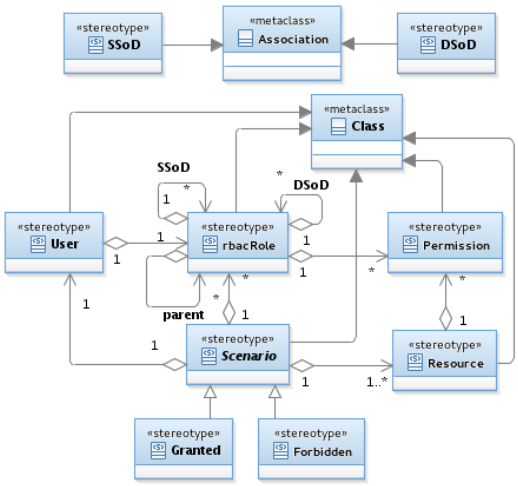

Fig. 7: DSML metamodel

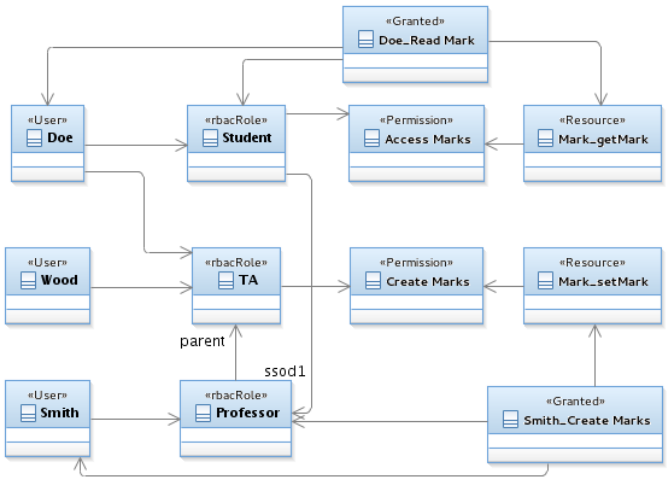

Fig. 8: Sample model with `rbacDSML`

TABLE I: OCL constraints for the DSML. WF = well-formedness, VAL = validation

Constraint	type
No SSoD violations	WF
No DSoD violations	WF
Activated roles are assigned to the user	WF
Granted scenarios are satisfied	VAL
Forbidden scenarios are satisfied	VAL

smaller, and hence easier and faster to fix automatically. We then proceeded to create the DSML.

C. The DSML

We developed the DSML from the same initial domain-specific metamodel. We did use UML in order to benefit from existing tool capabilities, and Selic's approach was used too. Fig. 7 shows the extension of the UML metamodel for this profile. All stereotypes are defined on UML classes, as it makes it easy to express assignments using associations. Associations between two roles are also stereotyped, to distinguish between SSoD, DSoD and role hierarchies.

Five OCL constraints complement the DSML: three make sure that the model itself is well formed, whilst the other two validate the model against the scenarios (see Table I). This is significantly less constraints than for the DSMAL, and they are also less complex and therefore faster to evaluate.

Fig. 8 is the same marking system example as Figs. 4, 5 and 6, but represented with the DSML. On the top left are the users, roles and permissions, and their assignments. On the top right are the two resources that are protected, `Mark_getMark` and `Mark_setMark`. At the bottom are two `Granted` scenarios. The first one requires that the user `Smith`, with the `Professor` role activated, must be able to access the `Mark_setMark` resource. The second one requires that the user `Doe`, with the `Student` role activated, must be able to access the `Mark_getMark` resource.

D. When to Use which Profile

Two things need to be considered when deciding which representation is the most appropriate for a particular activity: the activity itself, and the actor performing it. If the activity involves exclusively domain-specific concepts, then the DSML is preferable, as it is not cluttered with irrelevant constructs. If, however, general-purpose concepts also need to be considered, then the DSMAL is the only solution, because these do not appear on the DSML.

The actor is another important factor: a system administrator or a security expert may not be familiar with the UML notation, or simply not interested. They will likely prefer the DSML view. Developers, on the other hand, as well as designers and customers with technical expertise, may want to use the DSMAL to benefit from the context in which RBAC concepts are applied.

IV. Transformations

Both representations are derived from the same domain metamodel. Since the choice of profile to use depends on the circumstances, designers need transformations from one profile to the other, and back. It must be possible to reflect changes made to one representation on the other one, in order to keep both models synchronised. Since changes could happen in both representations, it is important to define a bidirectional transformation that would allow designers to transform a DSMAL model into a DSML, and back. Since the DSML contains a subset of the information contained in the DSMAL, tools such as GRoundTram [16] allow one to define a transformation from the DSMAL to the DSML, and the transformation from the DSML back to the DSMAL comes "for free". If both models can change before being synchronised, then the problem is similar to model-code synchronisation problems that MDE approaches face [17]. A similar approach to what is needed in this case is Guerra and

de Lara's use of triple graph transformations to derive views from models [18]. However, their views are not editable.

A. The Problem with DSMALs

The transformations we call for in this paper differ from transformations between "ad-hoc" DSMLs and DSMLs implemented as UML profiles, such as those described by Wimmer [10]. The latter are produced semi-automatically. Wimmer assumes that a transformation can "destroy" the target model and entirely replace it with the newly generated model. This is a fair assumption when dealing with DSMLs only, as there is no more information in one representation than in the other. But in the case of a DSMAL, such an approach will destroy all the information in the model that is not related to the DSML. In the example in Fig. 5, a destructive approach would completely erase the `Student`, `Module`, `Professor` and `TA` classes, as well as the `Mark::getDate()` and `Mark::setDate()` operations, in the class diagram (Fig. 5) alone. This problem would not occur with bidirectional transformations defined with GRoundTram, as it keeps track of the information that was removed during the forward transformation between the DSMAL and the DSML.

Yet, Wimmer's approach can still partially apply here, especially his discussion of the mapping between both languages. Since both languages stem from the same domain metamodel, it is not unlikely that many concepts will have a one-to-one mapping. However, the necessary redundancy in the DSMAL means that some mappings will be one-to-many, which further complicates the creation of the transformation.

B. Multiple Profiles

Annotating UML models with concepts from one domain is already challenging, but what if we want to annotate it with concepts from *several* domains? There may be many interesting ways of using DSMLs to model some aspects of the software: RBAC is one of them, but there are others, such as performance, specific business rules, persistence, etc. It would make sense to have a DSML as well as a DSMAL for each of them. The same general-purpose model could then be annotated with stereotypes from several profiles, which brings new challenges to keep the general-purpose model and the DSMLs in sync, and to prevent conflicts from happening.

1) Synchronisation Issues: If the same general-purpose model is annotated with concepts from several profiles, there is a chance that synchronisation problems will occur. Indeed, if a model is annotated with both access control and performance annotations, and changes are done concurrently to their respective DSML views, then the transformation that reflects these changes back to the general-purpose model will be more difficult. This problem is similar to the model-code synchronisation problem typically encountered in MDE [17].

2) Conflicts: It is possible that two DSMALs will bring conflicting annotations, and therefore a mechanism will be necessary to detect them. For example, let's assume two DSMALs, one for performance and one for persistence. If a particular element is marked with a performance requirement, and then marked with a persistence stereotype that involves saving the element's state in a database, the persistence will have a negative effect on the performance. These potential conflicts between stereotypes will have to be identified on a case by case basis. OCL constraints could be used to detect potential conflict and bring them to the designer's attention.

V. OCL Constraints

A. Where to Evaluate OCL Constraints

The distinction between DSML and DSMAL allows one to choose the best representation for the evaluation of OCL constraints. Indeed, the size of the model and the complexity of the constraints will influence the duration of the evaluation.

The `rbacUML` DSML only contains 5 OCL constraints to ensure well-formedness and conformance to the requirements. The DSMAL contains 5 of these constraints as well, that enforce the exact same properties, plus another 29 for consistency, because the same concept in the DSML can be translated into several concepts in the DSMAL. The 5 first constraints can be evaluated on the DSML: the model is smaller and the constraints are less complex, and hence the evaluation will be faster. On the other hand, the other 29 constraints must be evaluated on the DSMAL, as they simply do not exist on the DSML level. In `rbacUML`, we have developed another 4 categories of OCL constraints that provide further analysis on the model: scenario satisfiability, scenario coverage, model completeness and redundancy detection. These constraints can all be evaluated on the DSML representation because they are only concerned with the domain-specific concepts themselves, and thus one will benefit from faster evaluation times. Constraints that look at the boundary between the domain-specific concepts, or at how the domain-specific concepts integrate into the general-purpose model, must be evaluated on the DSMAL.

In `rbacUML` we are also developing automated correction of models. This helps the designers in finding a solution when well-formedness or verification constraints are violated. This is a very time- and resource-consuming process, and the computation time is greatly influenced by the size of the model and by the number of OCL constraints to satisfy. Indeed, the fixing engine tries to fix constraints one by one until a solution is found. The DSML, with its smaller size and smaller number of OCL constraints than the DSMAL, is a much more suitable choice, especially since the process is only concerned with RBAC-related concepts.

B. Translating OCL Constraints

In Section V-A we have argued that there is an ideal representation on which to evaluate OCL constraints. While this is true for constraints taken individually, it may still be needed to evaluate a constraint on the DSMAL even though the DSML would be a better fit, e.g. if the evaluation is part of a bigger analysis feature. Since the same domain-level properties need to be guaranteed, it is not unreasonable to expect that one could create their OCL constraints on the DSML, which is the simplest model and therefore requires relatively simple

constraints, and have them translated to be used with the DSMAL. After all, there is already a mapping between the concepts in the DSML and the concepts in the DSMAL.

However, the redundancy and the division of one concept into several ones will make this process more difficult. Not only is it necessary to create *new* OCL constraints to ensure consistency between redundant and spread out concepts, but the existing constraints may also have to take those into account, which would make them even more complex. An automated conversion process that would, given both profiles, the mapping of concepts from one to the other, and the OCL constraints for the DSML, produce the OCL constraints for the DSMAL, would be incredibly useful, as it would save time and reduce the occurrence of errors from the manual translation process. With GRoundTram, we could do this automatic conversion by giving a *filter- promotion* transformation for the transformation language UnQL+ [19]. Let T be the mapping from DSMAL to DSML in UnQL+ and C be a constraint on DSML. The filter promotion transformation is to transform $\mathit{filter}\ C \circ T$ to $T \circ \mathit{filter}\ C'$ to promote the condition C on the output of T to a new condition C' on the input.

In `rbacUML`, there is a one-to-one mapping for most concepts, which is relatively obvious from the two extensions of the UML metamodel. Three concepts, however, require a one-to-many mapping: the user, represented in the DSMAL on one class in the access control diagram and on any number of partitions in activity diagrams; the resources, represented in the DSMAL on an operation in a class diagram, and on any number of messages in a sequence diagram; the role activation, spread in the DSMAL over the `ActivateRoles` and `DeactivateRoles` on scenarios, and over partitions.

This makes translating the OCL constraints from the DSML to the DSMAL quite difficult. For example, the split of the user concept into several elements means that a new constraint must be introduced to make sure that all user annotations that are supposed to represent the same user are consistent, i.e. they must have the same name. This also makes the OCL constraint that checks that roles activated by a user are also assigned to him more complicated. In the DSML, the constraint looks like:

```
context rbacDSML::Scenario inv:
self.user.rbacRole
->includesAll(self.rbacRole)
```

but in the DSMAL, it looks like:

```
context rbacUML::Granted inv:
self.base_Action.inPartition
.extension_RBACUser.aliasUser.rBACRole
->includesAll(self.rBACRole)
```

Because of the separation of the user in two concepts, another OCL constraint is also necessary to ensure well-formedness:

```
context rbacUML::User inv:
(self.base_Partition <> null)
implies (self.name = self.user.name)
```

VI. Conclusion

In this position paper, we have argued for the need to further explore the representation of domain-specific concepts, and showed that both DSML- and DSMAL-based approach can be used together. We have discussed the place of OCL constraints evaluation, and argued that the choice of which representation to work with depends on the activity to be performed, the actor performing it as well as performance considerations. We have called for effective bidirectional transformations between both representations, and have also mentioned the issues that are likely to be caused by the use of several DSMALs on the same general-purpose language, turning it into a common repository from which domain-specific views of the model would be derived when necessary.

References

[1] *Unified Modeling Language (UML) 2.3*, OMG Std.

[2] B. Selic, "A systematic approach to domain-specific language design using UML," in *ISORC: Procs. Intl. Symp. Object and Component-Oriented Real-Time Distributed Computing*, may 2007, pp. 2 –9.

[3] *Object Constraint Language 2.2*, OMG Std.

[4] J. Bézivin, F. Jouault, and D. Touzet, "Principles, standards and tools for model engineering," in *ICECCS: Procs. Intl. Conf. on Engineering of Complex Computer Systems*. IEEE, 2005, pp. 28–29.

[5] 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).

[6] OMG, *Meta Object Facility (MOF) 2.0*, OMG Std.

[7] E. Fernández-Medina, J. Jurjens, J. Trujillo, and S. Jajodia, "Model-driven development for secure information systems," *Information and Software Technology*, vol. 51, no. 5, pp. 809 – 814, 2009, sPECIAL ISSUE: Model-Driven Development for Secure Information Systems.

[8] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security for process-oriented systems," in *SACMAT: Procs. Symposium on Access Control Models and Technologies*. ACM, 2003, pp. 100–109.

[9] A. Abouzahra, J. Bézivin, M. Del Fabro, and F. Jouault, "A practical approach to bridging domain specific languages with uml profiles," in *Procs. Best Practices for Model Driven Software Development at OOPSLA*, vol. 5, 2005.

[10] M. Wimmer, "A semi-automatic approach for bridging DSMLs with UML," *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 372 – 404, 2009.

[11] G. Giachetti, B. Marin, and O. Pastor, "Using UML profiles to interchange DSML and UML models," in *RCIS: Procs. Intl. Conf. on Research Challenges in Information Science*. IEEE, 2009, pp. 385 –394.

[12] M. H. Klein, *Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense Std. CSC-STD-001-83, August 1983, cSC-STD-001-83.

[13] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.

[14] "rbacUML tool," 2009-2012, http://computing-research.open.ac.uk/rbac/.

[15] IBM, "Rational Software Architect 8.0.4," 2012.

[16] S. Hidaka, Z. Hu, K. Matsuda, and K. Nakano, "Bidirectionalizing graph transformations," in *ICFP: Procs. Intl. Conf. on Functional Programming*. ACM, 2010, pp. 205–216.

[17] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, "Maintaining invariant traceability through bidirectional transformations," in *ICSE: Procs. Intl. Conf. on Software Engineering*. IEEE, 2012, pp. 540–550.

[18] E. Guerra and J. Lara, "Model view management with triple graph transformation systems," in *Graph Transformations*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4178, pp. 351–366.

[19] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "Towards a compositional approach to model transformation for software development," in *SAC: Procs. Symposium on Applied Computing*. ACM, 2009, pp. 468–475.