A Comparison of SecureUML and UMLsec for Role-based Access Control

Raimundas Matulevičius[1,2] and Marlon Dumas[1,2]

[1] Institute of Computer Science, University of Tartu,
J. Liivi 2, 50409 Tartu, Estonia
[2] Software Technology and Application Competence Center,
Ülikooli 8, 51003 Tartu, Estonia
{rma, dumas}@ut.ee

Abstract. Nowadays security has become an important aspect in information systems engineering. A mainstream method for information system security is Role-based Access Control (RBAC), which restricts system access to authorised users. Recently different authors have proposed a number of modelling languages (e.g., abuse cases, misuse cases, secure i*, secure Tropos, and KAOS extensions to security) that facilitate the documentation and analysis of security aspects. However it is unclear if these languages support the full spectrum of RBAC specification needs. In this paper we selected two security modelling languages, namely SecureUML and UMLsec. Based on the literature study and on the running example we systematically investigate how these languages could be used for RBAC. Our observations indicate that, although both approaches originate from the *de-facto* industry standard UML, they are not competitors. Rather they complement each other: SecureUML helps defining static RBAC aspects; UMLsec is recommended for dynamic RBAC analysis. Hopefully our study will help practitioners to understand these two approaches better, especially when selecting them for modelling purposes. We also believe that the combination of both approaches would ease secure information system development.

Keywords: Model-driven security, SecureUML, UMLsec, role-based access control, security modelling languages.

1 Introduction

Nowadays information systems play an important role in everybody's life. They are used in different areas and domains, including banking, education, medicine and others. People need to deal with information, which at many cases is confidential and should not be accessible for un-authorised use. Thus, ensuring security of information systems is a necessity rather than an option. Security is usually defined along four dimensions [3]: integrity (ensuring information is not altered), non-repudiation (ensuring receiving parties cannot renege on the receipt of information), authentication (confirming the originator and intended recipient of information) and confidentiality (ensuring information is shared only among authorised parties). In this paper, we focus

on the latter dimension and specifically, on one mechanism for ensuring confidentiality, namely Role-Based Access Control (RBAC) that restricts information access to authorised users.

Although security is an important aspect in information systems engineering, the literature [13], [23] reports that security concerns are often raised only when the system is about to be deployed or is already in use, or in the best-case security is considered only during the late system development stages (e.g., implementation). This is a serious hindrance to secure system development, since the early stages (e.g., requirements and design) are the place where system security concerns should be discovered and security trade-offs should be analysed. One possible way to guide such an analysis is suggested by the *model-driven security* approaches. For instance, Abuse frames [16] suggest means to consider security during the early phases of requirements engineering. Secure *i** [5] addresses security trade-offs. KAOS [15] was augmented with anti-goal models designed to elicit attackers' rationales. In [9] Tropos has been extended with the notions of ownership, permission and trust. Another version of Secure Tropos suggested in [20] models security using security constraints and attack methods. Abuse cases [19], misuse cases [22] and mal-activity diagrams [23] address security concerns through negative scenarios executed by the attacker.

All these modelling approaches could be applied to model RBAC in a system [1], however they are rather general than specific. In the literature we have observed that there are two modelling approaches – SecureUML [17] and UMLsec [12] – that, actually, contain targeted concepts for RBAC. Our motivation to look deeper at these two techniques was also strengthened by the fact that they both originate from UML, a *de facto* industry standard. Thus, we formulated the following research question:

> *What are the major similarities and differences between SecureUML and UMLsec for RBAC modelling?*

In order to answer this research question we have analysed the SecureUML and UMLsec literature and tested both approaches on a *Meeting Scheduler* example [7]. Our observations are that these approaches are not competitors when it comes to RBAC modelling, but they rather complement each other with different system modelling perspectives.

The structure of the paper is as follows: in Section 2 we introduce the general RBAC model and two modelling approaches – SecureUML and UMLsec. In Section 3 we compare SecureUML and UMLsec. Section 4 discusses our results and situates them in the state of the art. Finally, in Section 5 we conclude our study and present some future work.

2 Background

In this section we present the major artefacts discussed in this paper. Firstly, we recall the general RBAC model. Next, we present the major principles of SecureUML and UMLsec.

2.1 Role-based Access Control

The standard RBAC model is provided in [8] and displayed in Fig. 1. The main elements of this model are *Users*, *Roles*, *Objects*, *Operations*, and *Permissions*. A *User* is typically defined as a human being or a software agent. A *Role* is a job function within the context of an organisation. Role refers to authority and responsibility conferred on the user assigned to this role. *Permissions* are approvals to perform one or more *Operations* on one or more protected *Objects*. An *Operation* is an executable sequence of actions that can be initiated by the system entities. An *Object* is a protected system resource (or a set of resources). Two major relationships in this model are *User assignment* and *Permission assignment*. *User assignment* relationship describes how users are assigned to their roles. *Permission assignment* relationship characterises the set of privileges assigned to a *Role*.

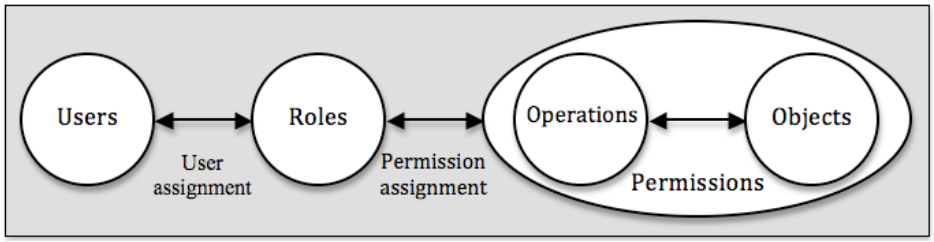

Fig. 1. Role-based Access Control Model (adapted from [8])

2.2 SecureUML

The SecureUML meta-model [2], [17] based on the RBAC model, is shown in Fig. 2. It defines the abstract syntax to annotate UML diagrams with information pertaining to access control. The meta-model introduces concepts like User, Role, and Permission as well as relationships between them. Protected resources are expressed using the standard UML elements (concept of ModelElement). In addition ResourseSet represents a user defined set of model elements used to define permissions and authorisation constraints.

The semantics of Permission is defined through ActionType elements used to classify permissions. Here every ActionType represents a class of security-relevant operations (e.g., *read*, *change*, *delete*, and etc) on a particular type of protected resource. On another hand a ResourceType defines all action types available for a particular meta-model type. An AuthorisationConstraint is a part of the access control policy. It expresses a precondition imposed to every call to an operation of a particular resource. This precondition usually depends on the dynamic state of the resource, the current call, or the environment. The authorisation constraint is attached either directly or indirectly, via permissions, to a particular model element representing a protected resource. The concrete syntax of SecureUML is illustrated in Fig. 3 and discussed in Section 3.2.

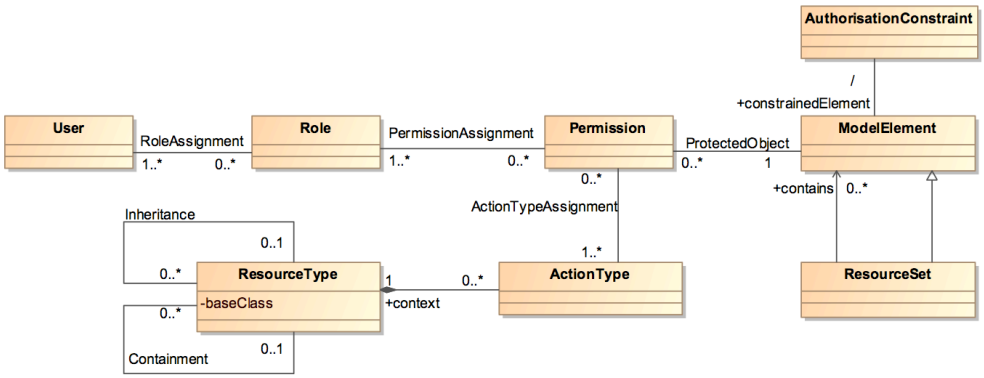

Fig. 2. SecureUML meta-model (adapted from [2], [17])

2.3 UMLsec

A major purpose of security modelling is to define mechanisms to satisfy security criteria, such as confidentiality and integrity [18]. To support this activity UMLSec [12], [13] is defined as a UML profile extension using stereotypes, tagged values and constraints (see Table 1). Constraints specify security requirements. Threat specifications correspond to actions taken by the adversary. Thus, different threat scenarios can be specified based on adversary strengths.

A subset of UMLsec that is directly relevant to this study are the role-based access control stereotype – **<<rbac>>** – its tagged values and constraints [13]. This stereotype enforces RBAC in the business process specified in the activity diagram. It has three associated tags **{protected}**, **{role}**, and **{right}**. The tag **{protected}** describes the states in the activity diagram, the access to whose activities should be protected. The **{role}** tag may have as its value a list of pairs (*actor*, *role*) where *actor* is an actor in the activity diagram, and *role* is a role. The tag **{right}** has as its value a list of pairs (*role*, *right*) where *role* is a role and *right* represents the right to access a protected resource. The associated constraint requires that the actors in the activity diagram only perform actions for which they have the appropriate rights. The application of the **<<rbac>>** stereotype is illustrated in Section 3.

3 Comparison

In order to compare SecureUML and UMLsec, firstly, we confront their general characteristics. Next, using the *Meeting scheduler* example, we illustrate how SecureUML and UMLsec could be used to develop RBAC models.

Table 1. UMLsec stereotypes (adapted from [12], [13]). The scope of this paper is highlighted in *italic*

Stereotypes	Base class	Tags	Constraints	Description
fair exchange	subsysem	start, stop, adversary	after start eventually reach stop	enforce fair exchange
rbac	*subsystem*	*protected, role, right*	*only permitted activities executed*	*enforces RBAC*
Intenet encrypted	link			Internet connection is encrypted
smart card	node			smart card node
critical	subsystem, object	secrecy, integrity, authenticity, high, fresh		critical object
data security	subsystem	adversary, integrity, authenticity	provides secrecy, integrity, authenticity, freshness	basic data security constraints
guarded access	subsystem		guarded object accessed through guards	access control using guarded objects
guarded	object	guard		guarded object

3.1 Literature-based Comparison of SecureUML and UMLsec

Our general SecureUML and UMLsec comparison is based on the SecureUML and UMLsec literature analysis. The primary sources are [17] for SecureUML and [12] for UMLsec. We selected these two because they correspond to the "time-stamp" of both approaches: they both are published at the same venue. However, for the sake of completeness we also consider two other sources: [2] for SecureUML and [13] for UMLsec. In this section we focus on the language extension mechanism, modelling targets and language application guidance. The results are summarised in Table 2.

Extension mechanism. Both approaches originate from the UML language[1]. SecureUML is developed through the explicit meta-model presented in Section 2. On another hand, UMLsec does not have an explicit meta-model, but the general UML meta-model [21] is implicitly extended with security modelling concerns.

Both SecureUML and UMLsec are proposed as the "lightweight UML extensions", namely new stereotypes, tagged values and constraints are defined. The difference is in the stereotype meaning. SecureUML is specifically oriented to the terminology of the RBAC model, and defines stereotypes, such as <<user>>, <<role>>,

[1] We base our analysis on a UML version available at a time [12] and [17] were published. With the development of UML 2.x the extension mechanism might be different. For instance UML 2.x does not contain explicit tagged values (these become additional attributes of stereotypes). However we did not observe any major UML changes, that would influence our analysis of SecureUML and UMLsec. Our comparison remains relevant independently of the UML version.

<<permission>>, <<actiontype>>, <<secureresource>>. This means that SecureUML extends mainly the profile of the UML class diagrams[2]. UMLsec takes a broader scope. It extends the whole UML profile, extending base classes such as model *subsystem*, *link*, *node*, *dependency*, and *object*. The extension stereotypes are applied at different UML diagrams. Hence, here the <<rbac>> stereotype is only one extension of the *subsystem* base class, applied in the *activity* diagram.

Authorisation constraints in SecureUML are written in the object constraint language (OCL) [24]. In UMLsec no specific constraint language is mandated.

Table 2. General comparison of SecureUML and UMLsec

	Criteria	SecureUML	UMLsec
Extension mechanism	**Meta-model**	Explicit, based on the RBAC model	Not explicit, as the UML profile extension
	UML profile	Mainly *class* diagrams	The whole UML profile (*use cases*, *class*, *activity*, *state*, *component*, and other diagrams)
	Extension mechanism	Stereotypes, tagged values and authentication constraints	Stereotypes, tagged values and constraints
	Constraints	Written in OCL	Constraint language is not identified
Modelling targets and application method	**Security criteria**	Not identified	Confidentiality, integrity, authenticity and other
	Security requirements	RBAC	RBAC, non-reputation, secure communication link, secrecy and integrity, authenticity, freshness, secure information flows, and guarded access
	Method	Development of the RBAC models	Not explicit, but implicitly supports standard security management methods

Modelling targets and application method. In the security risk management literature, a security criterion expresses a problem domain by "characterising the security needs" [18]. Security requirements describe a solution domain by defining a condition "we wish to make true by installing the system in order to mitigate risks" [18].

In our study we did not identify how SecureUML could model security criteria. SecureUML is basically meant for modelling of solutions especially through RBAC models. This results in SecureUML application guidelines, basically oriented towards RBAC development.

[2] In [16] authors speak about "host language" that would be he main language to model the system and the software itself. However, in the example the SecureUML authors primarily focus on class diagrams as the host language. Although some application of SecureUML stereotypes are proposed for state diagrams [2], the examples are limited and do not give much detail.

UMLsec is meant to perform a formal analysis of system security. The language could be applied at both problem and solution domains. In a UMLsec model, one can define confidentiality, integrity, and authenticity criteria. To satisfy the identified security criteria, UMLsec proposes solution stereotypes, such as security policies for a fair exchange, RBAC, non-repudiation, secure communication link, secrecy and integrity, authenticity, freshness, and secure information flow definitions. The application of UMLsec implicitly supports standard security risk management [4], [6], [10] that includes secure asset identification, definition of security criteria, risk analysis, and security requirements and controls definition.

3.2 RBAC Modelling using SecureUML and UMLsec

In this section we will consider a well-known *Meeting scheduler* example [7]. It is described as follows: *Meeting initiator* needs to organise a *top-secret* meeting. He needs to invite potential *Meeting participants* and find a suitable meeting *place* and *time*. In order to ease his task *Meeting initiator* decides to use a *Meeting scheduler system* for sending invitations, merging availability dates and informing the *Meeting participants*. Since the *Meeting* is top secret, the *Meeting scheduler system* must apply appropriate security policy for the *Meeting agreement* (*place* and *time*). This means, the *time* and *place* could be entered and changed only by the *Meeting initiator* and could be viewed only by the invited *Meeting participants*. In other words, no unintended audience should get access to the *Meeting agreement*. We will illustrate how this problem can be modelled with SecureUML and with UMLsec.

SecureUML. In Fig. 3 we present a SecureUML model to illustrate RBAC policy for the *Meeting Scheduler System*. Here we define three users Bob, Ann and John, who play different roles in the system. We also present that a resource (MeetingAgreement), which characterise *place* and *time* of the meeting, needs to be secured. Thus, a certain restriction on changing the state (changing the value of the attributes place and time) of this resource needs to be defined for the role MeetingInitiator and role MeetingParticipant.

Association class InitiatorPermissions characterises two *actions* allowed for the MeetingInitiator: (*i*) action enterAgreementDetails (of type Insert) defines that MeetingInitiator can enter *time* and *date* by executing operation setTimePlace() (see class MeetingAgreement), and (*ii*) action changeMeetingInfo (of type Update) allows changing *place* and *time* of the MeetingAgreement by executing operation changeTimePlace() (see class MeetingAgreement). To strengthen these permissions we define *authorisation* constraints AC#1 and AC#2:

AC#1:

```
context MeetingAgreement::setTimePlace():void
pre: self.roleInitiator.assignedUser ->
       exists(i | i.assignedUser = "Bob")
```

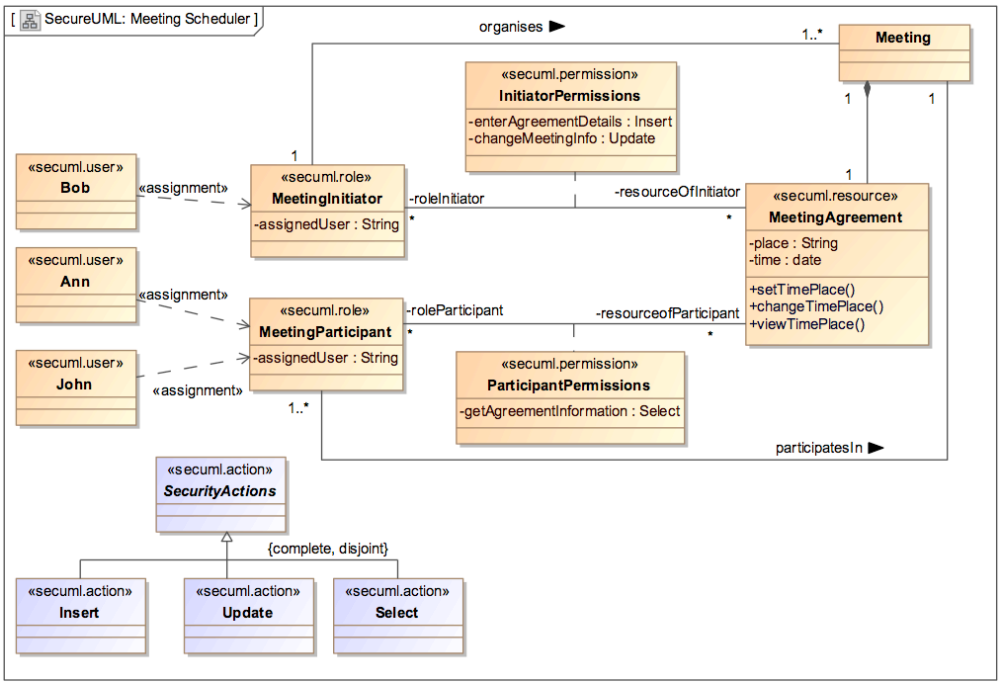

Fig. 3. Meeting Scheduler with SecureUML

Authorisation constraint AC#1 means that operation setTimePlace() (of class MeetingAgreement) can be executed (enter *time* and *place*), by one user *Bob* assigned to a role MeetingInitiator[3]. Similarly, the authorisation constraint AC#2 defines restriction for operation changeTimePlace() (of class MeetingAgreement):

AC#2:

```
context MeetingAgreement::changeTimePlace():void
pre: self.roleInitiator.assignedUser ->
       exists(i|i.assignedUser = "Bob")
```

Association class ParticipantPermissions defines a restriction for the MeetingParticipant role. It defines an *action* getAgreementInformation (of type Select) that says that only MeetingParticipant can view *place* and *time* defined in the MeetingAgreement. To enforce this permission an authorisation constraint AC#3 is defined:

[3] As illustrated in [2] SecureUML model might contain both objects (e.g., Bob, Ann, and John) and classes (e.g., MeetingInitiator, MeetingParticipant) in the same diagram when defining *user assignment* relationship. This results in authorisation constraints AC#1, AC#2 and AC#3 being very restrictive and allowing only the exact users defined in the model to perform operations on the secured resource (e.g., MeetingAgreement). In case the user assignment relationship was not specified, the precondition might be expressed like `self.roleInitiator.assignedUser=caller`, where `caller` is a set of users (assigned to a role) on behalf of whom the operation is executed.

AC#3:

```
context MeetingAgreement::viewTimePlace():void
pre: self.roleParticipant->
       exists (p1|p1.assignedUser="Ann") and
     self.roleParticipant->
       exists (p2|p2.assignedUser="John") and
     self.roleParticipant->size = 2
```

Authorisation constraint AC#3 says that only users *Ann* and *John* who have an assigned role MeetingParticipant can execute an operation viewTimePlace() (of class MeetingAgreement).

UMLsec. Fig. 4 illustrates application of UMLsec to model the *Meeting Scheduler System*. Here we define an activity diagram, which describes an interaction between MeetingInitiator, MeetingAgreement, and MeetingParticipant. The diagram specifies that MeetingInitiator can insert meeting time and date. Next MeetingParticipant is able to check if the time and place are suitable to him. If the agreement is not OK, MeetingParticipant requests MeetingInitiator to update. After the agreement (*time* and *date* of the meeting) data are updated MeetingParticipant can check them again for suitability.

This diagram carries an <<rbac>> stereotype, meaning that the security policy needs to be applied to the protected actions. For instance, the MeetingInitiator's action Insert meeting time and date leads to the action Set time and date for the MeetingAgreement. Set time and date is executed if and only if there exists an associated tag, that defines the following: (*i*) Set time and date is a protected action, (*ii*) *Bob* plays a role of MeetingInitiator, and (*iii*) MeetingInitiator enforces the action Set time and date. In the activity diagram this associated tag (AT#1) is defined as follows:

AT#1:

```
{protected = Set time and date}
{role = (Bob, MeetingInitiator)}
{right = (MeetingInitiator, Set time and date)}
```

Similarly, the sets of associated tags are defined for other two protected actions View time and date (AT#2) and Change time and date (AT#2). Note that both *Ann* and *John* can initiate execution of action View time and date (AT#2), since they both play the role of MeetingParticipant.

AT#2:

```
{protected = View time and date}
{role = ([Ann, John], MeetingParticipant)}
{right = (MeetingParticipant, View time and date)}
```

AT#3:

```
{protected = Change time and date}
{role = (Bob, MeetingInitiator)}
{right = (MeetingInitiator, Change time and date)}
```

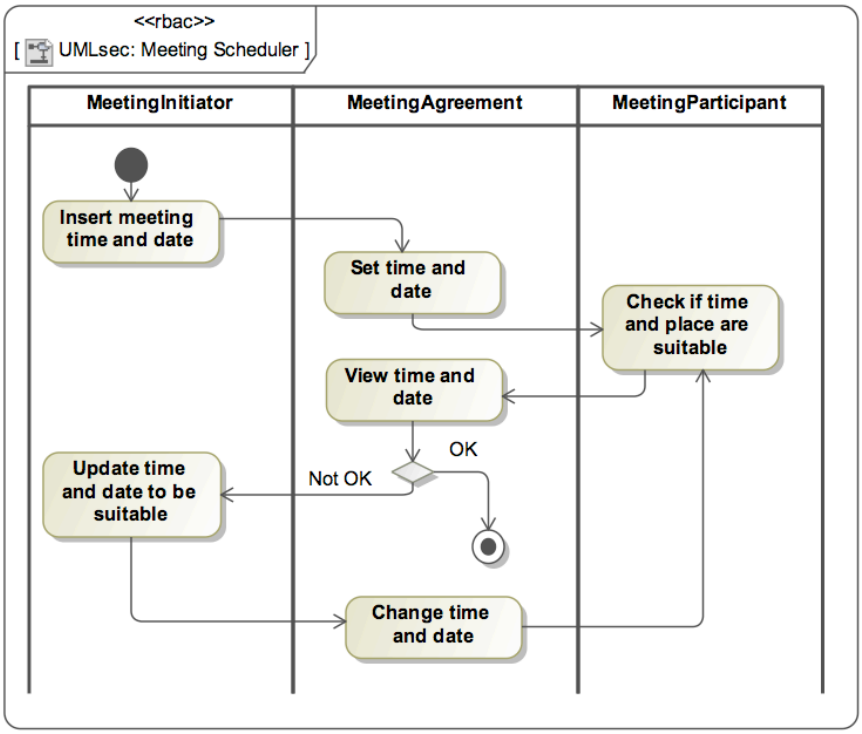

Fig. 4. Meeting Scheduler with UMLsec

Comparison of modelling constructs. In Table 3 we compare the RBAC modelling using SecureUML and UMLsec. We base our comparison on the RBAC model [8] presented in Section 2.1 (see Fig. 1). In this comparison we review which constructs are used for expressing the RBAC concepts and relationships. In Table 3 we also provide construct examples from Fig. 3 and Fig. 4.

Firstly, we observe that both approaches cover all RBAC concepts and relationships. This means that both approaches can express security policies through RBAC. Secondly, SecureUML addresses RBAC through the defined stereotypes, while than UMLsec expresses RBAC concepts and relationships through the associated tags and their values. At the example level, both approaches use the same (e.g., for *Users*, *Roles*, and *Objects*) or very similar (e.g., for *Operations*) labels for the RBAC concepts.

Some modelling and labelling differences are observed when it comes to relationship definition. In SecureUML *User assignment* relationship is modelled through a UML-stereotyped dependency without defining any label. In UMLsec the specific associated tag – {role} – is defined for a user-to-role assignment. Different labelling is also used to specify *Permission assignments*. In SecureUML this is done through stereotyped association classes, which might carry a name depending on the modelled context. In UMLsec the associated tag – {right} – is used for this purpose.

Finally, we note, that UMLsec does provide explicit means to define *Permission* itself. This is rather left implicitly at the diagram level when defining the values for all the associated tags ({protected}, {role}, and {right}). In SecureUML *Permissions* are explicitly defined through authorisation constraints expressed in OCL (see, for instance, the preconditions in AC#1, AC#2, and AC#3).

Table 3. Comparison of RBAC modelling using SecureUML and UMLsec

RBAC concepts	SecureUML		UMLsec	
	Construct	**Example**	**Construct**	**Example**
Users (concept)	Class stereotype <<secuml.user>>	Bob, Ann, and John	*Actor* value of the associated tag {role}	"Bob", "Ann", and "John"
User assignment (relationship)	Dependency stereotype <<assignment>>	Dependency between classes such as Bob and MeetingInitiator, and Ann or John and MeetingParticipant	Associated tag {role}	{role = (Bob, MeetingInitiator)} {role = ([Ann, John], MeetingParticipant)}
Roles (concept)	Class stereotype <<secuml.role>>	MeetingInitiator and MeetingParticipant	*Role* value of the associated tag {role}	"MeetingInitiator" and "MeetingParticipant"
Permission assignment (relationship)	Association class stereotype <<secuml. permission>>	InitiatorPermissions and ParticipantPermissions	Associated tag {right}	{right = (MeetingInitiator, Set time and date)} {right = (MeetingParticipant, View time and date)} {right = (MeetingInitiator, Change time and date)}
Objects (concept)	Class stereotype <<secuml. resource >>	MeetingAgreement	Activity partition	MeetingAgreement
Operations (concept)	Class operations	setTimePlace(), changeTimePlace(), and viewTimePlace()	An action	Set time and date, View time and date, and Change time and date
Permissions (concept)	Authorisation constraint	AC#1, AC#2, and AC#3	All three association tags {role} {protected}, and {right}	Not defined explicitely

4 Discussion and Conclusions

In this section, first, we will discuss limitations of our comparison. Next, we will see how our analysis confronts to the results found in the related work.

4.1 Limitations

This study is not without limitations. Firstly, we need to note that our analysis is of limited scope, as it is based on the literature survey and on a simple example (e.g., *Meeting Scheduler System* [7]). It might be the case that if we carried out an extensive

empirical study or a set of benchmarking examples we would receive different comparison results. Secondly, although we followed the theory as close as possible, our modelling example (in Fig. 3 and Fig. 4) carries a certain degree of subjectivity regarding the modelling decisions.

Thirdly, we should note that for our analysis we have selected only extracts of the modelling approaches. This is especially true for the UMLsec, where we focussed only on the means to define RBAC aspects. In addition to this UMLsec also provides other stereotypes (e.g., <<guarded access>> and <<guarded>>) that are used to deal with access control policies (without considering role assignments).

In the comparative example we skipped the application of UMLsec throughout the security risk management process [4], [6], [10]. For example, in Fig. 4 we could consider MeetingAgreement actions as *assets* and analyse the security criteria, such as *confidentiality*, *integrity*, and *availability* of the meeting agreement. Then, we could define a new swim-lane, which would illustrate how an *attacker* could exploit system *vulnerabilities* in order to break the security. However, in this paper we specifically focus on the definition of the security solution (through RBAC), because the goal is to contrast UMLSec with SecureUML, and the latter does not support security risk management.

4.2 Related Work

In the literature we identified two surveys [1], [11] that consider different security modelling approaches [5], [9], [12], [15], [16], [17], [19], [20], [22], [23] and their application for the RBAC modelling. In this section we specifically address the observation about SecureUML and UMLsec.

In [1] the $BRAC_0$ pattern is applied for comparison of security modelling approaches. The survey shows that, on one hand, SecureUML does not explicitly model security criteria (such as confidentiality, integrity, and availability) but it focuses on modelling the solutions to security problems guided by the RBAC nature. With SecureUML, a modeller can define assets, however, the language does not allow expressing attacks or harms to the assets. On the other hand, UMLsec is guided by security criteria, however it does not have means to model them explicitly. The UMLsec application is driven by analysis of system vulnerabilities: (*i*) once security vulnerabilities have been identified, the system design is progressively refined to eliminate the potential threats; (*ii*) the refinement of the design might be continued until the system satisfies the security criteria. Although UMLsec was analysed based on the $BRAC_0$ pattern, authors does not specifically indicate how well this approach is suitable for RBAC modelling.

In [11] Jayaram and Mathur investigate how the practice of software engineering blends with the requirement of secure software. The work describes a two-dimensional relationship between the software lifecycle stages and modelling approaches used to engineer security requirements. A part of the study is dedicated to RBAC modelling using SecureUML and UMLsec. Authors indicate that UMLsec is rather general approach than specific, thus it cannot be used to model access control policies solely. On another hand SecureUML is suggested as the means to specify access control

policies. However SecureUML cannot describe protected resources (system design), thus, it has to be used in conjunction with a base modelling language (e.g., ComponentUML).

Results of our study support findings of both [1] and [11]. For instance like in [1] we also notice the limitation of SecureUML to indicate security criteria. We also observe that the UMLsec application follows the standard security modelling methods (e.g., [4], [6], [10]). However our findings contradict those of [11] regarding UMLsec support for RBAC modelling. We observe that UMLsec provides means for RBAC modelling: it helps defining the dynamic characteristics of the secure system. Our analysis suggests that both modelling approaches can complement each other and result in more complete specifications of secure information systems (where both static and dynamic characteristics are defined).

5 Conclusions and Future Work

In this paper we have analysed how SecureUML and UMLsec can help defining security policies through the role-based access control mechanism. The contribution of this study is twofold. Firstly, a comparison of two security modelling languages gives modellers the criteria to selecting techniques for the RBAC analysis. Our major conclusions include the following:

- We observe that both SecureUML and UMLsec are applicable to model RBAC solutions. Table 3 illustrates that both approaches have means to address the RBAC concepts and relationships. The strong feature of SecureUML is the explicit definition of *Permissions* through authorisation constraints using OCL. On another hand our general comparison showed that at the methodological level SecureUML only focus on the solution domain. UMLsec provides means to identify and consider system risks, determine system vulnerabilities, and also develop solutions (RBAC is one of them) to mitigate the identified risks.
- Although both approaches originate from UML, SecureUML and UMLsec focus on different modelling perspectives to define security policies. SecureUML is used to model static characteristics of RBAC, thus, it is applied in the class diagrams. UMLsec is used to model dynamic characteristics of RBAC[4], thus it relies heavily on activity diagrams. Taking into account that system modelling needs to be addressed from different modelling perspectives [14], this means that both approaches are not competitors, but they rather complement each other by providing different viewpoints to the secure system.

In this paper we did not have a purpose to define model transformation rules between these approaches. However, a second part of our contribution is a set of guidelines (see Table 3) that could facilitate preparation of the RBAC activity diagram (using UMLsec) if we have a defined SecureUML class diagram or *vice versa* (SecureUML class diagram if the UMLsec activity diagram is done). But we also acknowledge that these guidelines, currently, should *not* be taken for granted because a further and more

[4] This is relevant only for the UMLsec <<rbac>> stereotype, other UMLsec stereotypes can be applied at different model types.

fine-grained analysis is necessary in order to define SecureUML and UMLsec transformation rules for RBAC. Such a definition remains for future work.

Acknowledgments. This research is funded by Logica and the European Regional Development Funds through the Estonian Competence Centre Programme. The authors would like to thank Andreas Sisask and Henri Lakk from Logica Estonia for the discussions and collaboration on this work. We also like to thank the anonymous reviewers whose comments helped to improve the camera-ready copy of this paper.

References

1. Bandara, A., Shinpei, H., Jurjens, J., Kaiya, H., Kubo, A., Laney, R., Mouratidis, H., Nhlabatsi, A., Nuseibeh, B., Tahara, Y., Tun, T., Washizaki, H., Yoshioka, N., Yu, Y.: Security Patterns: Comparing Modelling Approaches. Technical Report No 1009/06, Department of Computing Faculty of mathematics, Computing Technology, The Open University (2009)
2. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructure. ACM Transactions on Software Engineering and Methodology (TOSEM), 15 (1), 39--91 (2006)
3. Caelli, W., Longley, D., and Shain, M.: Information Security Handbook. Macmillan Publishers, 1991.
4. DCSSL. EBIOS - Expression of Needs and Identification of Security Objectives (2004)
5. Elahi, G., Yu, E.: A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs. In: Parent, C., Schewe, K.D., Storey, V.C., Thalheim, B. (eds.) Proceedings of the 26th International Conference on Conceptual Modelling (ER 2007) (2007)
6. ENISA. Inventory of Risk Assessment and Risk Management Methods (2004)
7. Feather, M.S., Fickas, S., Finkelstein, A., van Lamsweerde A.: Requirements and Specification Exemplars. Automated Software Engineering, 4: 419--438 (1997)
8. Ferraiolo D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-based Access Bontrol. ACM Transactions on Information and System Security (TISSEC), 4(3), 224--274 (2001)
9. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling Security Requirements Through Ownership, Permision and Delegation. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05), IEEE Computer Society (2005)
10. ISO. Information technology - Security techniques - Information security management systems - Requirements, International Organisation for Standardisation (2005)
11. Jayaram, K.R., Mathur, A.P.: Software Engineering for Secure Software – State of the Art: a Survey. Technical report CERIAS TR 2005-67, Department of Computer Sciences & CERIAS, Purdue University (2005)
12. Jurjens, J.: UMLsec: Exending UML for Secure Systems Development. In: Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460, pp. 412--425. Springer-Verlag (2002)
13. Jurjens. J.: Secure Systems Development with UML. Springer-Verlag Berlin Heidelberg, (2005)
14. Krogstie J., Solvberg A.: Information Systems Engineering, Conceptual Modeling in a Quality Perspective. Course compendium, Norwegian University of Science and Technology, (2000)

15. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-models. In: Proceedings of the 26th International Conference on Software Engineering (ICSE'04), IEEE Computer Society pp. 148--157 (2004)
16. Lin, L., Nuseibeh, B., Ince, D., Jackson, M.: Using Abuse Frames to Bound the Scope of Security Problems. In: Proceedings of the 12th IEEE international Conference on Requirements Engineering (RE'04), IEEE Computer Society (2004) 354--355
17. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-based Modeling Language for Model-driven Security. In: Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460, pp. 426--441. Springer-Verlag (2002)
18. Mayer N.: Model-based Management of Information System Security Risk. PhD Thesis, University of Namur (2009)
19. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In: Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99). 55 (1999)
20. Mouratidis, H.: Analysing Security Requirements of Information Systems using Tropos. In: Proceedings 1st Annual Conference on Advances in Computing and Technology (AC&T), London - United Kingdom, pp. 55--64 (2006)
21. OMG, Unified Modeling Language: Superstructure, version 2.0 (2005)
22. Sindre, G.: Mal-activity Diagrams for Capturing Attacks on Business Processes. In: Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007), Springer-Verlag Berlin Heidelberg, pp. 355--366 (2007)
23. Sindre, G., Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering Journal 10(1) pp. 34--44 (2005)
24. Warner, J., Kleippe, A.: The Object Constraint Language, second edition, Getting Your Models Ready for MDA. Addison-Wesley (2003)