Towards Model Transformation between SecureUML and UMLsec for Role-based Access Control

Raimundas MATULEVIČIUS a, b, 1 and Marlon DUMAS a, b

a Institute of Computer Science, University of Tartu, J. Liivi 2, 50409 Tartu, Estonia
b Software Technology and Application Competence Center, Ülikooli 4, 51003 Tartu, Estonia

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 R*, 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. Thus, the combined use of both approaches would provide a more comprehensive approach to secure information system development. As a step towards enabling the combined use of SecureUML and UMLsec, this paper outlines a mapping transformation between these two languages.

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

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 [1]: 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 [2], [3] 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 [4] suggest means to consider security during the early phases of requirements engineering. Secure i* [5] addresses security trade-offs. KAOS [6] was augmented with anti-goal models designed to elicit attackers’ rationales. In [7] Tropos has been extended with the notions of ownership, permission and trust. Another version of Secure Tropos suggested in [8] models security using security constraints and attack methods. Abuse cases [9], misuse cases [3] and mal-activity diagrams [10] address security concerns through negative scenarios executed by the attacker.

All these modelling approaches could be applied to model RBAC in a system [11], however they are rather general than specific. In the literature we have observed that there are two modelling approaches – SecureUML [12][13] and UMLsec [2][14] – that, actually, contain targeted concepts for RBAC. In [15] we report on a comparison of SecureUML and UMLsec according to the literature study and some illustrative example [16]. Our observations are that these approaches are not competitors when it comes to RBAC modelling, but they rather complement each other by addressing different security modelling perspectives. Therefore, an opportunity exists to employ these approaches in combination, so as to obtain more comprehensive information systems security models, particularly with respect to RBAC. As a first step towards enabling a combined use of SecureUML and UMLsec, we outline a set of rules for transforming a SecureUML model to an UMLsec model and vice versa. In this way, developers can start with a model in either of these approaches and later on, enrich this initial model using the other approach.

The structure of the paper is as follows: in Section 1 we introduce the general RBAC model and two modelling approaches – SecureUML and UMLsec. In Section 2 we compare SecureUML and UMLsec using the illustrative example. Based on the comparison results we define a set of semi-automatic transformation rules in Section 3. Finally, in Section 4 we conclude our study and present some future work.

1. Background

In this section we present the major artefacts discussed in this paper. Firstly, we recall the general RBAC model [17]. Next, we present the major principles of SecureUML and UMLsec.

1.1. Role-based Access Control

The main elements of the RBAC model [17] 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.

1.2. SecureUML

The SecureUML meta-model [12] [13] is based on the RBAC model. It defines the abstract syntax (see Figure 1) 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 ResourceSet represents a user defined set of model elements used to define permissions and authorisation constraints.

![SecureUML meta-model](adapted from [12], [13])

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 Figure 2 and discussed in Section 2.1.

1.3. 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 [2]
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.

Table 1. UMLsec stereotypes (adapted from [2][14]). The scope of this paper is highlighted in italic

<table>
<thead>
<tr>
<th>Stereotypes</th>
<th>Base class</th>
<th>Tags</th>
<th>Constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fair exchange</td>
<td>subsystem</td>
<td>start, stop, adversary</td>
<td>after start eventually reach stop</td>
<td>enforce fair exchange</td>
</tr>
<tr>
<td>Rbac</td>
<td>subsystem</td>
<td>protected, role, right</td>
<td>only permitted activities executed</td>
<td>enforces RBAC</td>
</tr>
<tr>
<td>Internet encrypted</td>
<td>link</td>
<td></td>
<td>Internet connection is encrypted</td>
<td></td>
</tr>
<tr>
<td>smart card</td>
<td>node</td>
<td></td>
<td>smart card node</td>
<td></td>
</tr>
<tr>
<td>critical</td>
<td>subsystem, object</td>
<td>secrecy, integrity, authenticity, high, fresh</td>
<td>critical object</td>
<td></td>
</tr>
<tr>
<td>data security</td>
<td>subsystem</td>
<td>adversary, integrity, authenticity</td>
<td>provides secrecy, integrity, authenticity, freshness</td>
<td>basic data security constraints</td>
</tr>
<tr>
<td>guarded access</td>
<td>subsystem</td>
<td></td>
<td>guarded object accessed through guards</td>
<td>access control using guarded objects</td>
</tr>
<tr>
<td>guarded</td>
<td>object</td>
<td>Guard</td>
<td>guarded object</td>
<td></td>
</tr>
</tbody>
</table>

A subset of UMLsec that is directly relevant to this study is the role-based access control stereotype – <<rbac>> – its tagged values and constraints [2]. 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 2.2.

2. Comparison

In [15] we compare SecureUML and UMLsec based on the literature study. Like in [11], there we notice the limitation of SecureUML to indicate security criteria. We also observe that the UMLsec application follows the standard security modelling methods [18]. In contradiction to [19] we observe that UMLsec provides means for RBAC modelling. Thus, we suggest that both approaches can complement each other and result in more complete specifications of secure information systems.

In this work we compare modelling constructs of SecureUML and UMLsec following the Meeting scheduler example [16], which 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.

### 2.1. SecureUML Model

In Figure 2 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:

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

![Diagram](image.png)

**Figure 2.** 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*. Similarly, the authorisation constraint AC#2 defines restriction for operation `changeTimePlace()` (of class *Meeting Agreement*):

```java
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:

```java
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*).

### 2.2. UMLsec Model

Figure 3 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:

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

---

2 As illustrated in [12] SecureUML model might contain both objects (e.g., *Bob*, *Ann*, and *John*) and classes (e.g., *MeetingInitiator*, *MeetingParticipant*). This results in restrictive authorisation constraints AC#1, AC#2 and AC#3. 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.
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*.

\[
\text{AT#2} \quad \text{protected} = \{\text{View time and date}\} \\
\quad \text{role} = \{(\text{Ann, John}), \text{MeetingParticipant}\} \\
\quad \text{right} = \{\text{MeetingParticipant, View time and date}\}
\]

\[
\text{AT#3} \quad \text{protected} = \{\text{Change time and date}\} \\
\quad \text{role} = \{\text{Bob, MeetingInitiator}\} \\
\quad \text{right} = \{\text{MeetingInitiator, Change time and place}\}
\]

Figure 3. Meeting Scheduler with UMLsec

2.3. Comparing SecureUML and UMLsec Constructs

In Table 2 we compare the RBAC modelling using SecureUML and UMLsec. We base our comparison on the RBAC model [17] presented in Section 2.1. In this comparison we review which constructs are used for expressing the RBAC concepts and relationships. In Table 2 we also provide construct examples from Figure 2 and Figure 3.

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 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.

Table 2. Comparison of RBAC modelling using SecureUML and UMLsec

<table>
<thead>
<tr>
<th>RBAC concepts</th>
<th>SecureUML</th>
<th>UMLsec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Construct</td>
<td>Example</td>
</tr>
<tr>
<td>1 Users (concept)</td>
<td>Class stereotype (&lt;\text{secuml}\text{.user}&gt;&gt;)</td>
<td>Bob, Ann, and John</td>
</tr>
<tr>
<td>2 User assignment (relationship)</td>
<td>Dependency stereotype (&lt;\text{assignment}&gt;&gt;)</td>
<td>Dependency between classes such as Bob and MeetingInitiator, and Ann or John and MeetingParticipant</td>
</tr>
<tr>
<td>3 Roles (concept)</td>
<td>Class stereotype (&lt;\text{secuml}\text{.role}&gt;&gt;)</td>
<td>MeetingInitiator and MeetingParticipant</td>
</tr>
<tr>
<td>4 Permission assignment (relationship)</td>
<td>Association class stereotype (&lt;\text{secuml}\text{.permission}&gt;&gt;)</td>
<td>Operations of association classes InitiatorPermissions and ParticipantPermissions</td>
</tr>
<tr>
<td>5 Objects (concept)</td>
<td>Class stereotype (&lt;\text{secuml}\text{.resource}&gt;&gt;)</td>
<td>MeetingAgreement</td>
</tr>
<tr>
<td>6 Operations (concept)</td>
<td>Operations of (&lt;\text{secuml}\text{.resource}&gt;&gt;) class</td>
<td>setTimePlace(), changeTimePlace(), and viewTimePlace()</td>
</tr>
<tr>
<td>7 Permissions (concept)</td>
<td>Authorisation constraint</td>
<td>AC#1, AC#2, and AC#3</td>
</tr>
</tbody>
</table>

Finally, we note, that UMLsec does not 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).

3. Transformation Rules

Following our comparative analysis of SecureUML and UMLsec for the RBAC we have developed a set of rules for transforming SecureUML models to UMLsec models
and vice versa. Some of these rules could be directly implemented in the modelling tools, thus resulting in the semi-automated support for RBAC model transformation. In order to illustrate our proposal we will incrementally show how the transformation rules would be applied to the SecureUML class diagram (e.g., Meeting Scheduler example in see Figure 2) to translate it to the UMLsec activity diagram (given in Figure 4).

3.1. Model Transformation from SecureUML to UMLsec

Below we define four transformation rules to transform a model from SecureUML to UMLsec:

SU1. A class with a stereotype <<secuml.resource>> is transformed to an activity partition in the UMLsec model (Table 2, line 5), and the operations of this class become actions belonging to this partition (Table 3, line 6). In addition, each operation becomes a value the UMLsec associated tag {protected}. Example: the class MeetingAgreement (see Figure 2) is represented as activity partition in Figure 4. The operations setTimePlace(), changeTimePlace(), and viewTimePlace() are shown as actions in this partition. Three associated tags (see Table 2) {protected} are defined: {protected=(setTimePlace)}, {protected=(changeTimePlace)}, and {protected=(viewTimePlace)};

SU2. A relationship with a stereotype <<assignment>> relationship used to connect users and their roles is transformed to an associated tag {role}. Example: From Figure 2 we specify associated tags {role=(Bob, MeetingScheduler)}, {role=(Ann, MeetingParticipant)}, and {role=(John, MeetingParticipant)}, as provided in Table 2.

SU3. A SecureUML class with the stereotype <<secuml.roles>> is transformed to the UMLsec activity partition (Table 2, line 3). The attributes of an association class that connects the <<secuml.roles>> class with <<secuml.resource>> class, become actions in the corresponding activity partition (Table 2, line 4). Example: in Figure 4 we define activity partitions MeetingInitiator (with the actions enterAgreementDetails and changeMeetingInfo grabbed from the class InitiatorPermission) and MeetingParticipant (with the action getAgreementInformation defined from the class ParticipantPermissions);

SU4. The SecureUML association class with the stereotype <<secuml.permission>> defines the role value for the UMLsec associated tag {right} (Table 2, line 4). The value of right can be determined from the authorisation constraint defined for the attribute of the SecureUML association class. Example: this rules leads to the association tags {right=(MeetingInitiator, setTimePlace)} and {right=(MeetingParticipant, viewTimePlace)} captured in Table 3.

Note: The authorisation constraint might help to identify the relationship between two actions as shown in Figure 4 between actions enterAgreementDetails and setTimePlace, and between actions getAgreementInformation and viewTimePlace.

Note: Complete definition of the associated tad {right} is not always possible because (i) it might be that no association constraint is defined at all (no additional security enforcement is needed), or (ii) an authorisation constraint might be
defined at a different place (e.g., to strengthen the operations of the <<secuml.resource>> classes) in the SecureUML model as shown for the authorisation constraint AC#2 in Figure 3.

<table>
<thead>
<tr>
<th>Meeting Initiator</th>
<th>MeetingAgreement</th>
<th>MeetingParticipant</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterAgreementDetails</td>
<td>setTimePlace</td>
<td>getAgreementInformation</td>
</tr>
<tr>
<td>changeMeetingInfo</td>
<td>changeTimePlace</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewTimePlace</td>
</tr>
</tbody>
</table>

Figure 4. Transformed UMLsec model (only automatic translations according to rules SU1-SU4 are shown)

In Table 3 we summarise the resulting association tags. However, as mentioned above, we are not able to determine all the association tags required in the model. For example the (right) association tag is captured from the association class <<secuml.permission>> and its link to the authorisation constraint. However not all association constraints are links to the association classes <<secuml.permission>>. For example, in Figure 2 the authorisation constraint AC#2 is linked to class MeetingAgreement. This means that the association tag \{right = (MeetingInitiator, changeTimePlace)\} needs to be written manually by the developer. This means that manually two actions need to be linked by a control flow as well (e.g., from ChangeMeetingInfo to changeTimePlace), given that SecureUML is not concerned with behavioural aspects.

Table 3. Automatically translated association tags

<table>
<thead>
<tr>
<th>Transformation rule</th>
<th>Association tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>SU1.</td>
<td>{protected=(setTimePlace)}</td>
</tr>
<tr>
<td></td>
<td>{protected=(changeTimePlace)}</td>
</tr>
<tr>
<td></td>
<td>{protected=(viewTimePlace)}</td>
</tr>
<tr>
<td>SU2.</td>
<td>{role=(Bob, MeetingScheduler)}</td>
</tr>
<tr>
<td></td>
<td>{role=(Ann, MeetingParticipant)}</td>
</tr>
<tr>
<td></td>
<td>{role=(John, MeetingParticipant)}</td>
</tr>
<tr>
<td>SU4.</td>
<td>{right =(MeetingInitiator, setTimePlace)}</td>
</tr>
<tr>
<td></td>
<td>{right =(MeetingParticipant, viewTimePlace)}</td>
</tr>
</tbody>
</table>

To complete the UMLsec activity diagram (e.g., Figure 4) a developer needs to specify information that was not possible to capture from the SecureUML diagram. For instance the developer needs to define initial node (e.g., to enterAgreementDetails action) and activity final node (e.g., from viewTimePlace action). Other control flows (including the conditionals ones) needs also to be specified. For instance in Figure 4 control flows between setTimePlace and getAgreementInformation, viewTimePlace and changeMeetingInfo, and changeTimePlace and getAgreementInformation might define a logical sequence of activity that corresponds to the one in Figure 3.
3.2. Model Transformation from UMLsec to SecureUML

The transformation from UMLsec to SecureUML is defined by means of four rules. To illustrate these rules, we will analyse the Meeting Scheduler example presented in Figure 3. The resulting SecureUML class diagram is given in Figure 5.

US1. Association tags {protected} allow us identify the operations that belong to secured resource (Table 2, line 6). We transform the activity partitions, which hold these operations to the SecureUML class with a stereotype <<secuml.resource>> (Table 2, line 5).

Example: in Figure 5 we define a class MeetingAgreement which has three operations Set time and place(), View time and place(), and Change time and place().

US2. In the UMLsec model the activity partitions that do not hold secured protected actions, can be transformed to the <<secuml.role>> classes (Table 3, line 3).

Example: In Figure 5 we define two <<secuml.role>> classes: MeetingInitiator and MeetingParticipant.

US3. Association tag {roles} allows us to identify the <<assignment>> dependency relationship (Table 2, line 2) between classes of users defined with a stereotype <<secuml.user>>, and their roles presented with a stereotype <<secuml.role>>.

Example: in Figure 5 we define three <<assignment>> dependency links: (i) between Bob and MeetingInitiator, (ii) between Ann and MeetingParticipant, and (iii) between John and MeetingParticipant.

US4. From UMLsec association tag {right} we are able to identify on which operations a role can perform security actions (Table 2, line 4). Thus, from each occurrence of this association tag in the SecureUML model, a corresponding association class between a <<umlsec.role>> and a <<umlsec.resource>> is introduced.

Example: In Figure 5 we define two association classes: (i) between MeetingInitiator and MeetingAgreement, and (ii) between MeetingParticipant and MeetingAgreement.

Furthermore, from the UMLsec activity diagram (Table 2, line 4) we are able to identify what exact security actions are carried towards the secured operations: these are unprotected actions performed before the protected ones.

Example: in Figure 3 the action Insert meeting time and place is performed before protected action Set time and place. We transform the Insert meeting time and place action to the attribute of the association class between MeetingInitiator and MeetingAgreement (see Figure 5). The similar transformations are performed for the attribute Update time and place to be suitable for the same association class, and for the attribute Check if time and place are suitable for the association class between MeetingParticipant and MeetingAgreement.

Note: we are not able to identify the type of security actions.

The SecureUML model needs to be completed manually with the information, which is not captured from the UMLsec model. Specifically, the developer needs to introduce the following information:

• the attributes of the <<secuml.resource>> class that define the state of the secured resource(s). For example, in Figure 5 the class MeetingAgreement should be complemented with attributes place:String and time:String.
all the necessary authorisation constraints for the SecureUML model. For instance for Figure 5 to correspond Figure 2 the authorisation constraints AC#1, AC#2, and AC#3 have to be defined;

* multiplicities for all the association relationships. For example, multiplicities for associations (see Figure 5) between MeetingInitiator and MeetingAgreement, MeetingParticipant and MeetingAgreement have to be defined;

* names for the association classes. For instance in Figure 5 names for classes with the <<secuml.permission>> stereotype have to be specified;

* action types for the identified actions. For example, for action Insert meeting time and place action type is Insert, for Update time and place to be suitable action type is Update, and for Check if time and place are suitable action type is Select.

Finally, the developer should also include additional model attributes as required. For instance in order the translated model (Figure 5) would correspond to the Meeting Scheduler in Figure 2, one needs to define attributes assignedUser:String for classes MeetingInitiator and MeetingParticipant. These attributes would carry the information about the users assigned for these roles.

Figure 5. Transformed SecureUML model
(only automatic translations according to rules US1-US4 are shown)

3.3. Threats to Study Validity

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 (e.g., [12], [13] for SecureUML and [2] [14] for UMLsec) and on a simple example (e.g., Meeting Scheduler System [16]). 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 and different rules for transformation. Secondly, although we followed the theory as close as possible, our modelling example (in Figure 2 and Figure 3) 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 our comparative example we skipped the application of UMLsec throughout the security risk management process [18]. For example, in Figure 3 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 activity partition, 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. 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. We observe that both SecureUML and UMLsec are applicable to model RBAC solutions. Table 2 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 focuses on the solution domain. Meanwhile, UMLsec provides means to identify and to consider system risks, to determine system vulnerabilities, and also to develop solutions to mitigate the identified risks (RBAC being one of such solutions) [15].

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 on top of class diagrams. UMLsec is used to model dynamic characteristics of RBAC\(^3\), thus it relies heavily on activity diagrams. Taking into account that system modelling needs to be addressed from different modelling perspectives, this means that both approaches are not competitors, but they rather complement each other by providing different viewpoints to the secure system.

In this work we outline a set of transformation rules in order to perform a semi-automatic translation from SecureUML to UMLsec and vice-versa. These rules can be implemented in a CASE tool in order to support a multi-perspective approach to RBAC definition. The implementation and empirical testing of the transformation rules currently remains a future work. Another direction for future work is to define an incremental transformation that would allow developers to maintain pairs of concurrently evolving SecureUML and UMLSec models synchronised.

\(^3\) This is relevant only for the UMLsec <<rbac>> stereotype, other UMLsec stereotypes can be applied at different model types.
Acknowledgment

This research is funded by Logica and the European Regional Development Funds through the Estonian Competence Centre Programme.

References