UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Liis Jaks

A Prototype for Transforming Role-Based Access Control Models

Bachelor's Thesis

Supervisors: Dr. Raimundas Matulevičius
Dr. Sven Laur

Author: ..... "....." May 2012
Supervisor: ..... "....." May 2012
Supervisor: ..... "....." May 2012
Approval for Defence
Professor: ..... "....." May 2012

TARTU 2012

Abstract

Role-based access control is a widely-used mechanism in computer systems – it ensures security by restricting resource access to only the system users with respective rights. The RBAC solutions can be engineered with the aid of modelling languages, such as SecureUML and UMLsec, which both present the system design from different viewpoints. Creating multiple coherent models, however, may turn out to be a non-trivial and time-consuming task. This, in turn, may dramatically lessen the motivation to create role-based access control models altogether.

As a solution to the problem above, developers could be provided a software tool, which inputs a model in one language and transforms it into the model of another. The transformed model, however, would not be complete, since the two languages are used to represent somewhat different information. The aim of such a tool would be to diminish the necessity to manually copy information, when creating a second model.

With this thesis, a prototype tool is developed, which enables the transformation of a SecureUML model to a UMLsec model and *vice versa*. The tool is implemented in the Java programming language, as a plug-in to the professional UML modelling tool MagicDraw. Menu items are added to the application, which trigger transformations: information is collected from a model in the UMLsec or SecureUML language and, based on that, a new model in the other language is created. As an additional function, completion checks are developed for both models to inform the user of whether all necessary language elements are present. They should act as guides for the user on how to improve the transformed model, since after transformations some information is known to be absent from the new model. Another additional component is the support for manipulating UMLsec association tags, which are an integral part of transformations between the SecureUML and UMLsec languages. The documentation – requirements, code documentation and user manual – is also provided in this paper and are supposed to contribute to the further development as well as understanding of the prototype.

Table of Contents

Appendices

List of Tables and Figures

Chapter 1: Introduction

Role-based access control (abbreviated RBAC) [1] is one of the major mechanisms to ensure information system and technology security. It guarantees that only the people who have the rights for information could access and manipulate it using these rights. However, engineering RBAC solutions is not a trivial task. It requires different viewpoints and involves different stakeholders.

Modelling languages, such as SecureUML and UMLsec, suggest concepts for RBAC definition using various models, typically based on UML (Unified Modelling Language), such as class diagrams in SecureUML and activity diagrams in UMLsec. These two languages enable taking different viewpoints into account when designing system security. Nevertheless, aligning different models to a coherent view still requires time from developers. One solution to the problem would be to offer the developer a tool, which support a model transformation from one language to another. The aim of the transformation function is to lessen the need for manual activities. Nevertheless, after the transformation the developer would need to manually add the elements that carry new information in the output language.

The purpose of this thesis is to develop a prototype tool for RBAC model transformation. This prototype is implemented as a plug-in to a well-accepted industrial modelling toolkit MagicDraw (version 17.0, developed and maintained by NoMagic)[1]. It is a closed-source product, which offers a wide API (Application Programming Interface) in the Java programming language. The API provides the possibility to create different application-specific plug-ins (e.g. a new menu item in the diagram context menu) by extending special-purpose classes. There are possibilities for programming activities that include reading and modifying the information of existing models and creating new ones. The tool developed consists of two transformations (from SecureUML to UMLsec and *vice versa*). Additional components are developed to increase the usability of the tool: support for manipulating UMLsec association tags and completion checks that generate guidelines on how to finish a model in either language.

The thesis is structured as follows. The first part introduces the background – in Chapters 2, 3 and 4 the concept of role-based access control, the security modelling languages SecureUML and UMLsec, and the modelling tool MagicDraw UML are discussed correspondingly. The second part of the thesis gives the details of the implemented prototype (see Chapter 5). This part also includes the requirements and design (see Chapter 6), structure of the program code (see Chapter 7), and a user manual (see Chapter 8). Additionally, the work contains appendices, which are provided on a separate CD. See Appendix A for code documentation in the Javadoc standard and Appendix B for the prototype code. Appendix C contains the files necessary for installation (see the user manual in Chapter 8 for installation guidelines). Sample models for testing can be found in Appendix D.

[1] See https://www.magicdraw.com/

Chapter 2: Role-Based Access Control

Ever since computer systems began to serve multiple users, it also became necessary to pay attention to new related security issues. Role-based access control is one of the means system administrators can use to ensure that only authorised people get access to resources [1]. The basic construct of RBAC is a role. Roles are created based on job functions performed in the company or organisation using the computer system. Specific permissions for accessing resources regarding their nature and extent of use allowed are grouped together to create roles. Then, users are assigned roles based on their job responsibilities and authority [1].

Depending on the configuration by the system administrator, RBAC can be easily used for creating settings that satisfy the three major security principles. The first is *least privilege*: a role is assigned the least set of privileges necessary for completing tasks by the user in that role. The second principle is *separation of duties*, the use of mutually exclusive roles. The last principle met by RBAC is *data abstraction*, which means the definition of data operations with an abstract meaning (e.g. credit and debit for an account object) [1].

2.1 Concept of RBAC

In this work we adapt the core RBAC model [1]. The model defines a minimum set of concepts and relationships in order to completely define a role-based access control system. The basic concept of RBAC is that *users* and *permissions* are assigned to *roles* and *users* acquire *permissions* by being members of *roles*. The same *user* can be assigned to many *roles* and a single *role* can have many *users*. Similarly, for *permissions*, a single *permission* can be assigned to many *roles* and a single *role* can be assigned to many *permissions*. The core RBAC model is shown in Figure 1. It includes five major concepts: *Users*, *Roles*, *Objects*, *Operations*, and *Permissions*. In addition, the Core RBAC model includes a set of *Sessions* where each session is a mapping between a user and an activated subset of roles that are assigned to the user.

A *User* is any person who interacts with a computer system, but this concept could also be extended to machines, networks, or intelligent autonomous agents. A *Role* is a job function within the context of an organisation. Some associated semantics include the authority and responsibility conferred on the user assigned to the role. *Permission* is an approval to perform an operation on one or more protected objects. An *Operation* is an executable image of a program, which upon invocation executes some function for the user. *User assignment* and *permission assignment* are characteristics of the set of privileges assigned to a role.

Each *Session* is a mapping of one *User* to possibly many *Roles*. This allows for the selective activation and deactivation of roles assigned to the user. The *Session roles* relationship defines the roles activated by the session. The *User sessions* relationship defines the set of permissions assigned to the roles that are activated across the user's session.

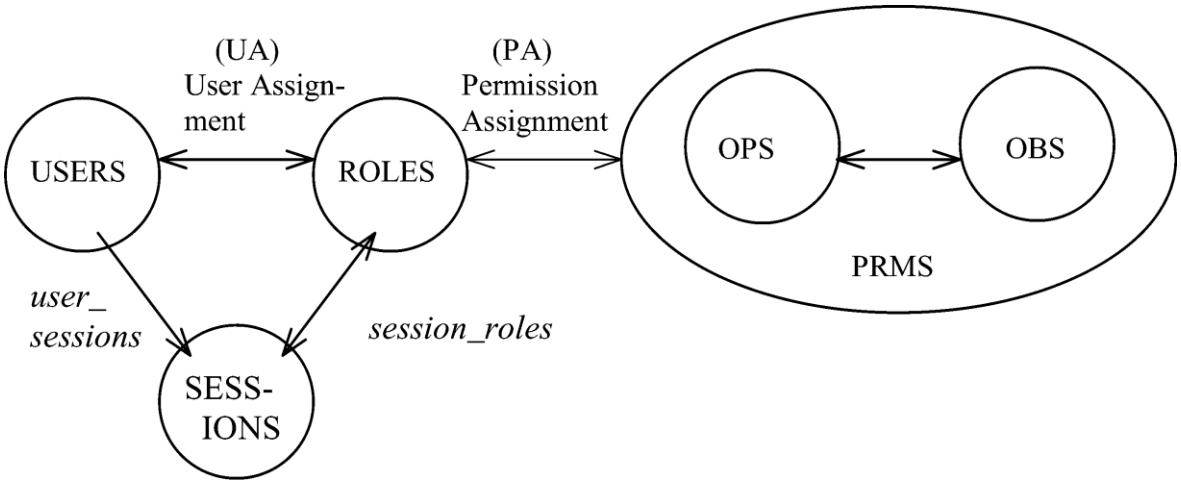

Figure 1. Core RBAC [2]

2.2 Further Components of RBAC

The first more advanced component of RBAC is *Hierarchical RBAC*, which adds requirements for supporting role hierarchies. Role hierarchy means a seniority relation between roles, whereby senior roles acquire the permissions of their juniors, and junior roles acquire the user membership of their seniors. Hierarchical RBAC becomes effective in settings where roles tend to overlap, meaning that common permissions may be assigned to users in different roles. It may also be time-consuming to repeatedly specify general permissions that are performed by a large number of users in an organisation.

Secondly, *Static Separation of Duty Relations* are defined. They are used to enforce policies against users gaining authorisation for permissions associated with conflicting roles. Generally, it means constraining the assignment of users to roles – for example, by enforcing a requirement that two roles are mutually exclusive (e.g. a role that requests expenditures *versus* a role that approves them). They can also be enforced, keeping the concept of role hierarchies in mind.

A similar component is described as *Dynamic Separation of Duty Relations*. DSD requirements limit the availability of permissions to a user by placing constraints on the roles that can be activated within or across the user's sessions. Therefore, a user is allowed to be authorised for roles that do not cause a conflict of interest when acted on independently, but may do so when activated simultaneously.

2.3 Summary

The core RBAC model defines the basic elements of any role-based access control system: *Users*, *Roles*, *Objects*, *Operations*, *Permissions* and *Sessions*. *Hierarchical RBAC* adds the requirement for role hierarchies - seniority relations between roles. *Static Separation of Duty Relations* require that two roles, which include mutually conflicting permissions, cannot be assigned to a user. They can also be applied, taking role hierarchies into consideration. *Dynamic Separation of Duty Relations* are a step forward

from *SSD*, in that roles, which include conflicting permission, can be assigned to a user, but not within the same session.

So far, the research of RBAC model transformations has not addressed the advanced components previously mentioned. The research, as well as the transformation tool prototype developed within this thesis, focuses solely on Core RBAC. The reason is that it describes all the features minimally necessary in a system that implements role-based access control [4], [5].

Chapter 3: Security Modelling Languages

This thesis uses two security modelling languages: SecureUML and UMLsec. In this chapter we will present them in an illustrative example – the *Meeting Scheduler* system [6], which envisages the basic application of role-based access control.

3.1 SecureUML

SecureUML is a security modelling language, the purpose of which is modelling RBAC solutions, rather than security criteria, and is meant to help with RBAC development. An example of a SecureUML diagram is presented in Figure 2. It uses UML class diagrams to depict the relationship between users, roles, permissions and resources. UML stereotypes help create the necessary language primitives by sub-typing UML core types [3]. The extension mechanism also includes tagging values with authorisation constraints for strengthening permissions, which are defined in the object constraint language (OCL) [4].

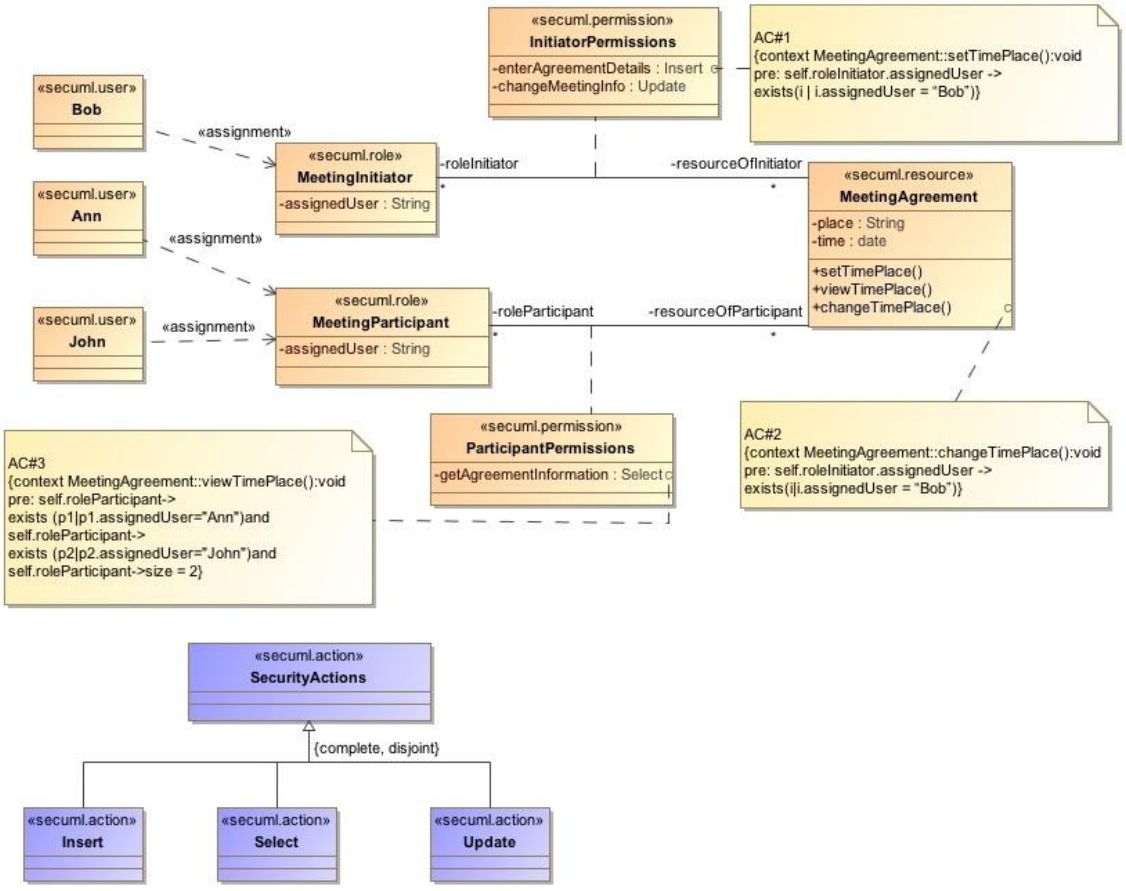

Figure 2. Meeting Scheduler with SecureUML [4]

The fragment of UML used within this thesis and its meaning in the context of SecureUML models is described as follows. Classes with the stereotypes <<securml.user>>, <<securml.role>> and <<securml.resource>> are used to portray SecureUML users, roles and resources respectively. Users are assigned to roles by dependency links, stereotyped <<securml.assignment>>.

Between roles and resources lie association classes with stereotypes <<securml.permission>>. The attributes of such an association class designate the actions that the users of the respective role allowed to perform on the resource. The types of those attributes are chosen from a collection of ActionTypes, elements to classify permissions: *read*, *change*, *delete*, *insert*, etc. [4]. Permissions can be constrained with dynamic aspects (e.g. time) to hold only in certain system states by tagging the association class's attributes with textual authorisation constraints [3].

The model features three association classes. AC#1 and AC#3 define the operations of the protected resource that can be performed by the users mentioned when they are assigned to the specified role. AC#2 defines a similar restriction for an operation of the protected resource.

3.2 UMLsec

UMLsec is an extension of the whole UML profile for broad security criteria analysis, security risk management and security requirements definition. Only a fragment of it is used for the purposes of this thesis – activity diagram bearing the stereotype <<rbac>> and textual association tags. An example of a UMLsec diagram is presented in Figure 3. The activity partitions represent protected resources and roles. Their actions stand for actions that can be performed on the resources or by the actors, respectively.

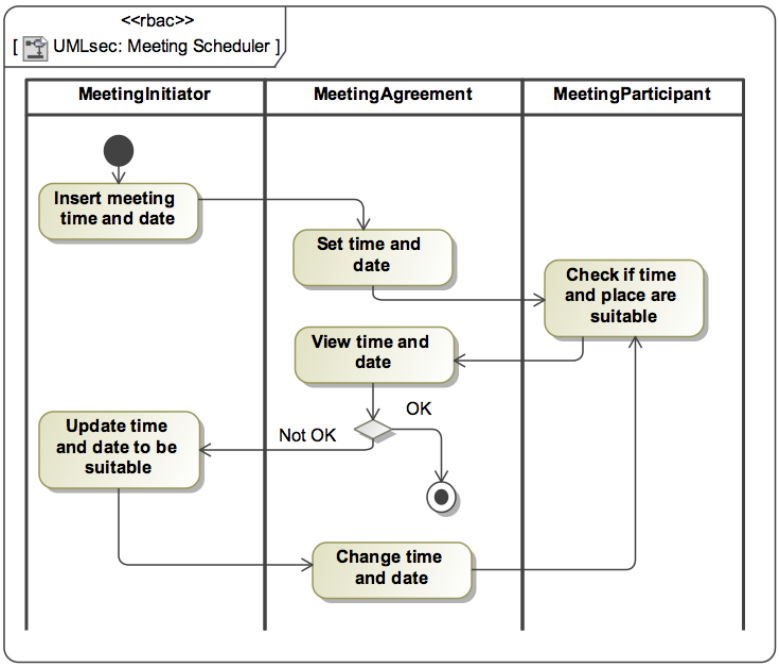

Figure 3. Meeting Scheduler with UMLsec [4]

The purpose of association tags is to define access control rules. They are presented in sets of three: {protected}, {role} and {right}. The {protected} tag represents an activity of a resource partition in the activity diagram to which access should be protected. The value of the {role} tag is a pair of an actor name and a role name, the latter of which is also present in the diagram as an activity partition. The {right} tag comprises the names of the previously stated role and protected action [4].

3.3 Transformation Rules

There are similarities in what information about a system's RBAC design is displayed in the SecureUML and UMLsec models (see Table 1). Therefore it is possible to translate some information presented in one language to information in the other. The requirements (see Chapters 6.2 and 6.3) for the development of the transformation plug-in are based on rules developed in [5]. The transformation rules are illustrated by the Meeting Scheduler example and presented in the exact form as done by Matulevičius and Dumas [5]. As can be expected, since the models are meant to carry different information, some elements need to be added to the transformed diagram manually by the developer.

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

Table 1. Comparison of RBAC modelling using SecureUML and UMLsec [4]

3.3.1 Model Transformation from SecureUML to UMLsec

In the following list, transformation rules are presented for model transformation from SecureUML to UMLsec [5], in a step-by-step structure. Requirements for the transformation, which have been compiled based on these rules, can be found in Chapter 6.2. For an illustration of what the outcome of a transformation by these rules should look like, see Figure 4.

SU1. A class with a stereotype <<securml.resource>> is transformed to an activity partition in the UMLsec model, and the operations of that class become actions belonging to this partition. In addition, each operation becomes a value of the UMLsec association tag {protected}.

SU2. A relationship with the stereotype <<assignment>> used to connect users and their roles is transformed to an association tag {role}.

SU3. A SecureUML class with the stereotype <<securml.roles>> is transformed to the UMLsec activity partition. The attributes of the association class that connects the <<securml.role>> class with a <<securml.resource>> class, become actions in the corresponding activity partition.

SU4. The SecureUML association class with the stereotype <<securml.permission>> defines the *role* value for the UMLsec association tag {right}. The value of *right* can be determined from the authorisation constraint defined for the attribute of the SecureUML association class.

Note: The authorisation constraint might help to identify the relationship between two actions.

Note: Complete definition of the association tag {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 <<securml.resource>> classes) in the SecureUML model, as shown for the authorisation constraint AC#2 in Figure 2.

Some things will remain undefined by the transformation. The developer still needs to add elements such as control flows, the initial node and the final node to the generated model.

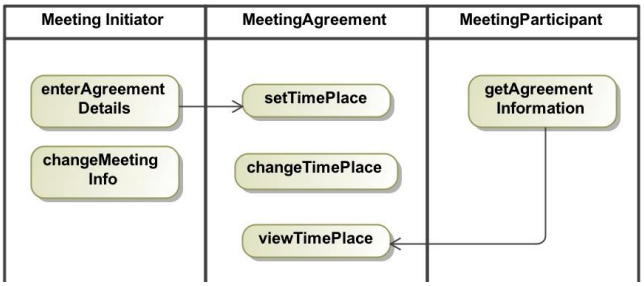

Figure 4. Theoretical picture of a transformed UMLsec model [5]

3.3.2 Model Transformation from UMLsec to SecureUML

In the following list, transformation rules are presented for model transformation from UMLsec to SecureUML [5], in a step-by-step structure. Requirements for the transformation, which have been compiled based on these rules, can be found in Chapter 6.3. For an illustration of what the outcome of a transformation by these rules should look like, see Figure 5.

US1. The association tags {protected} allow us to identify the operations that belong to a secured resource. We transform the activity partitions, which hold these operations to SecureUML classes with the stereotype <<secuml.resource>>.

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

US3. The association tags {role} allow us to identify the <<assignment>> dependency relationship between classes of users defined with the stereotype

<<secuml.user>>, and their roles presented with the stereotype <<secuml.role>>.

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

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).
- all the necessary authorisation constraints for the SecureUML model.
- multiplicities for all the association relationships.
- names for the association classes.
- action types for the identified actions.

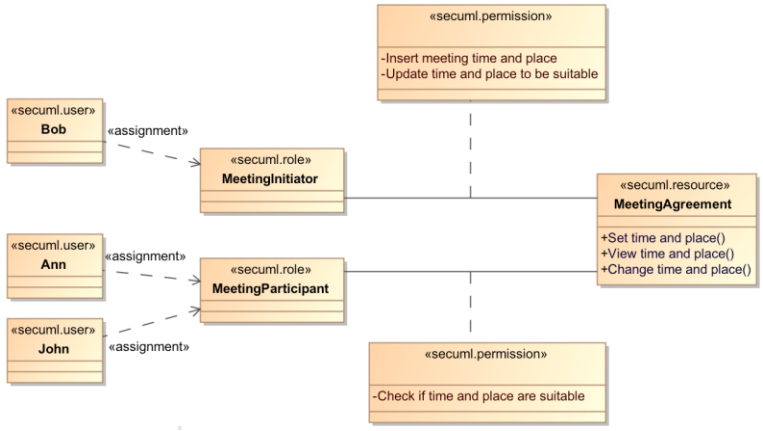

Figure 5. Theoretical picture of a transformed SecureUML model

3.4 Summary

There are a few limitations to these transformation rules. Firstly, they are based on only one example (the Meeting Scheduler System) and therefore it is possible that a more extensive empirical study could produce different results. The subjectivity of modelling decisions should also be taken account in this example.

It should be noted that only some extracts of the modelling approaches have been selected for the analysis, especially from UMLsec. It provides quite a few additional stereotypes besides <<rbac>> for dealing with access control. The application of UMLsec throughout the security risk management process has also been skipped in the interests of contrasting UMLsec with SecureUML – therefore only the definition of the security solution through RBAC has been focused on [4].

The prototype developed in this thesis will follow the previously presented steps for model transformations (see requirements analysis in Chapters 6.2 and 6.3). UMLsec association tags are included in a significant proportion of these rules, but they are not directly supported in MagicDraw, the modelling toolkit to which the prototype is developed as a plug-in. Therefore, a component for working with association tags is also developed (see requirements analysis in Chapter 6.1).

Chapter 4: MagicDraw UML

MagicDraw is a UML modelling and CASE tool created by No Magic, Inc. In the interest of this thesis, the company has given access to an API that can be used for developing plug-ins for the program. In the following section structural presentation of models in MagicDraw is outlined. Following that, the OpenAPI used for developing the plug-in is described. Information about using the OpenAPI has been provided in a respective user manual and Javadoc documentation that can be found on the MagicDraw web page[2]. A copy of the manual also comes with the installation, along with Javadoc files and numerous sample plug-ins.

4.1 Structure of Models

Regarding the structure of models in MagicDraw (see Figure 6), the top entity where all model data is held, is a project. Different diagrams of the model and their elements can be organised in packages. Each element can carry various properties and be connected to other elements with the aid of relationships. All model elements that are present in the project can be seen in a browser menu. Each element has an *owner*, which can be a package, a subsystem or some other appropriate element.

Although tightly connected, they are not the same entities as the elements visually seen on the diagram, which are called *symbols*. Model elements, such as a package, class, state, use case, object or other, can be notated on a diagram as a *shape*. The term *path* is used for the notation of relationships, such as associations, aggregations, dependency, message and links.

4.2 Development of Plug-ins

The MagicDraw OpenAPI libraries are present in the lib folder of the installation and are named *md_api.jar* and *md_common_api.jar*. Including them is necessary for creating any plug-in for the tool. The minimum requirement for a plug-in is that there needs to be a class derived from *com.nomagic.magicdraw.plugins.Plugin*. The functionality of the plug-in should be written in the *init()* method of that class.

In order to add the created plug-in to the program, one simply needs to create a subfolder under the *MagicDraw UML/plugins* folder with a few necessary files. Firstly, it needs to contain compiled java files of the plug-in code, packaged into a jar file. Secondly, a plug-in descriptor file is required. The name of the file has to be *plugin.xml* and should provide information about the plug-in id, name, version, provider, name of the class which extends the *com.nomagic.magicdraw.plugins.Plugin* class and the *init()* method of which needs to be called on start-up; requirements for starting the plug-in, and the jar library name. Additional files needed by the plug-in should also be present in the folder.

[2] See https://www.magicdraw.com/download_manual#openapi

The OpenAPI user manual is fairly detailed with respect to creating some certain types of additions, but does not cover all general types of actions a programmer would like to have the plug-in perform. However, it does give enough information to get the overall sense of how the architecture is organised. The provided example plug-ins may happen to cover some things that cannot be found in the manual, but they are not exhaustive either. While the Javadoc is compact by format and helpful for working with the API, one may find the descriptions of methods uninformative. The three sources of information are of significant help, but given the scope of the OpenAPI, difficulties may arise when having very specific functionality in mind for a plug-in to be created.

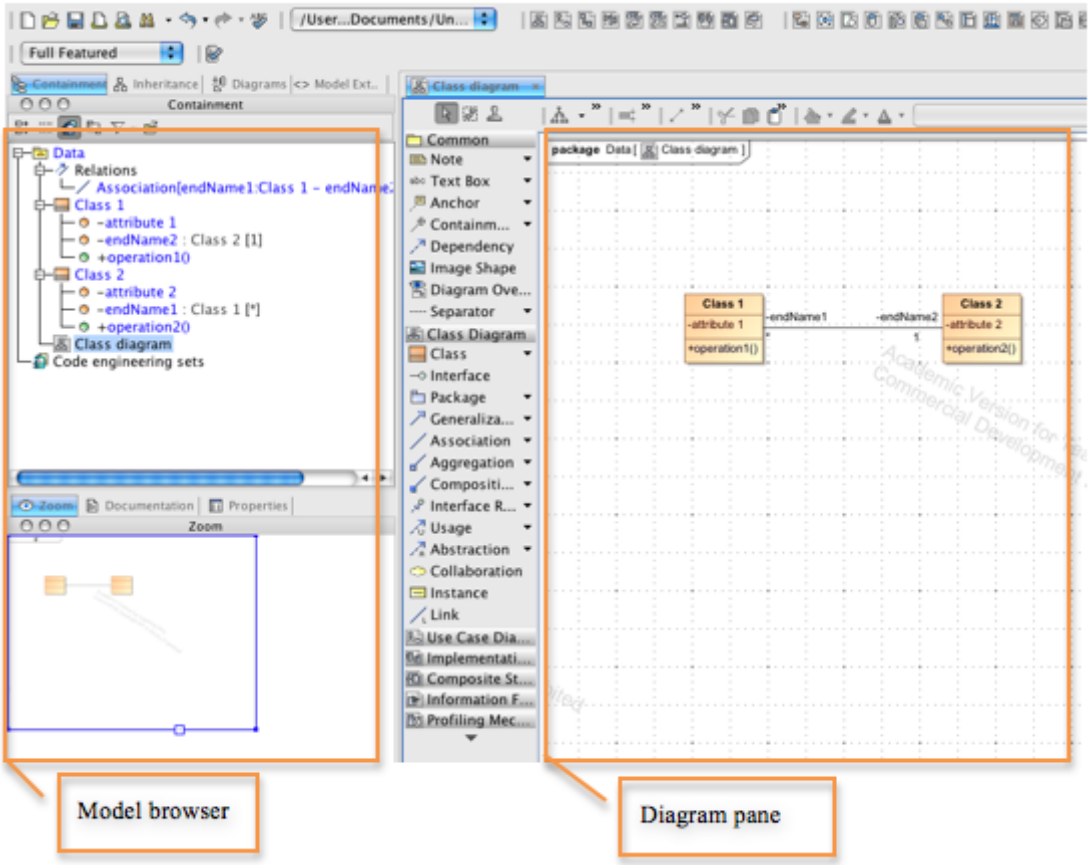

Figure 6. User interface of MagicDraw UML

4.3 Summary

MagicDraw was chosen as the program where to implement the transformations mainly for two reasons. For one, it is a widely acknowledged professional modelling tool. Secondly, its vendors have given the author of this thesis exclusive permission to work with the program's API. The application's level of detail and the possibilities provided by the API are suitable with the purposes of this thesis, enabling free low-level manipulation of models.

Chapter 5: Prototype

The aim of this thesis is to develop a prototype tool for security model transformations. The requirements analysis for the tool is based on previous studies on the possibilities of security model transformations [4] [5]. To host the prototype as a plug-in, the professional modelling tool MagicDraw UML has been chosen. For the purpose of creating this prototype, NoMagic Inc. has given permission to use its OpenAPI. The products of this thesis (see Figure 7) feature a tool for performing the transformations, its code and documentation.

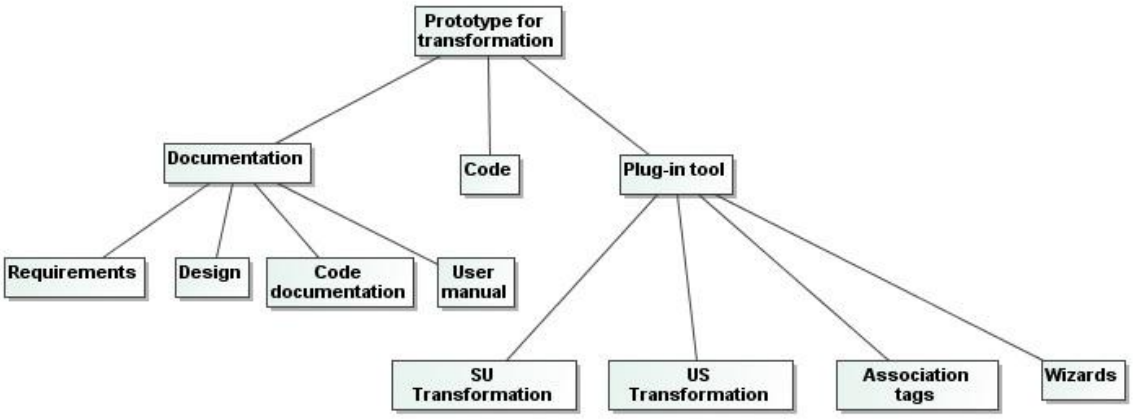

Figure 7. Prototype breakdown structure

The plug-in tool has components that fall into three categories:

- Transformations: translating a SecureUML model to a UMLsec model and *vice versa* ("SU Transformation" and "US Transformation" on Figure 7).
- Completion checks: the user can run them any time when constructing either one of the models ("Wizards" on Figure 7). They focus on elements that cannot be derived during the transformations and their purpose is to guide the user through completing the models.
- Association tags: a component for viewing, adding and editing UMLsec association tags ("Association tags" on Figure 7).

The code of the plug-in created (fully presented in Appendix B) is written in the Java programming language, since this is the language that the OpenAPI of MagicDraw UML is available in. Firstly, the aid of the OpenAPI manual was used to create the context menu elements for initialising transformations, association tags view and wizards. From there on, coding included describing the functionality of each action and close collaboration with the OpenAPI documentation was necessary to understand how to create, modify or read information from diagram elements.

The documentation includes the following elements:

- Requirements analysis (see Chapter 6): based on the transformation rules presented in Chapter 3.3. It is divided into three components: two transformations and support for UMLsec association tags. They have been compiled as detailed step-by-step instructions for following through each process.
- Design documents (see Chapter 7): includes automatically generated class diagrams of the code and descriptions of the code's structure.
- Javadoc (see Appendix A): code documentation is in HTML format and generated using the Javadoc tool. The code also includes additional comments, which are not visible in the Javadoc documentation, for further development of the plug-in. To view the documentation, open *index.html* in the *Appendix A – Javadoc/doc* folder on the attached CD.
- User manual (see Chapter 9): graphically describes how to use the transformation tool and its different features.

Chapter 6: Requirements and Design

Requirements for the plug-in are based on the transformation rules presented in Chapter 3.3. We expect the user to have created a SecureUML or UMLsec diagram that is to be transformed. MagicDraw has built-in tools for creating constructs that are necessary in these languages. It does not, however, feature the possibility to systematically manipulate SecureUML authorisation constraints and UMLsec association tags. The support for viewing and editing association tags on a UMLsec model is developed within this thesis. The transformations, completion checks and the association tags view are activated upon clicking respective menu items in the diagram context menu.

6.1 Support for UMLsec Association Tags

The following requirements describe the three main functionalities of the association tags component. A summary of actions that can be performed within this component is presented on Figure 8. Firstly, the user can open a window that displays all tags. From there they may move on to editing the existing tags or adding new ones.

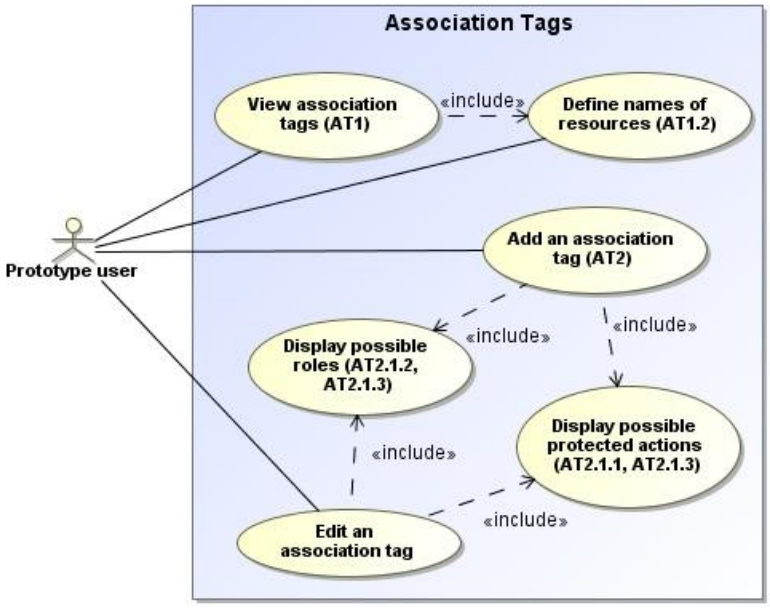

Figure 8. Use case diagram of association tags

AT1: Displaying

The requirements for displaying association tags, which are present on the model, are as follows:

AT1.1 The UMLsec association tags window can be opened upon clicking „UMLsec association tags" in the diagram context menu.

AT1.2 In the uppermost panel, display a list of checkboxes with the names of all activity partitions of the UMLsec model. The names checked signify protected resources.

AT1.2.1. If a name is unchecked, see if any actions from that partition are presented in the „protected =" part of any association tag. If yes, display an error message to the user and do not uncheck the name.

AT1.3 Present the association tags in a list. Next to each association tag display an „Edit" button (see **AT3**) and a „Delete" button

AT1.3.1. Clicking the „Delete" button will remove the association tag from the list

AT1.4 Below the association tags, display the buttons „Add" (see **AT2**), „Cancel" and „OK".

AT1.4.1. Clicking the „Cancel" button will restore the state of the association tags to the state that was before opening the association tags window.

AT1.4.2. Clicking the „OK" button will store the new state of the association tags.

AT2: Adding

The requirements for adding new association tags to the model are as follows:

AT2.1. Upon clicking the „Add" button, close the display window and open the adding window. Display three rows: „protected =", „role =" and „right =".

AT2.1.1. Next to „protected =" display a dropdown menu with all the actions that belong to the activity partitions, whose names were checked in **AT1.2** (by default, the first item in the menu is selected). (Incomplete – all actions are displayed and an error message is given when the wrong action is chosen.)

AT2.1.2. Next to „role =" display a text box (for the user name; empty by default) and a dropdown menu with all the names of the activity partitions, whose names were not checked in **AT1.2**. (by default, the first item in the menu is selected). (Incomplete – the names of all activity partitions are present and an error message is given when the wrong name is chosen.)

AT2.1.3. Next to „right =" display a dropdown menu identical to the one in **AT2.1.2**. (*role*). The two menus should be linked – if a menu item is chosen in one, the same item needs to be chosen in the other. Next to that, display a dropdown menu identical to the one in **AT2.1.1**. (*protected resource*). These two menus should be linked similarly.

AT2.1.4. Below the editing area, display three buttons: „Clear", „Cancel" and „OK".

AT2.2. Clicking the „Clear" button restores the default values of all fields.

AT2.3. Clicking the „Cancel" button will not add the association tag, close the adding window and open the association tags display window.

AT2.4. Clicking the „OK" button will add the association tag to the list, close the adding window and open the association tags display window.

AT3: Editing

The requirements for editing existing association tags are as follows:

AT3.1. Upon clicking the „Edit" button next to an association tag, the display window is closed and an identical window to **AT2.1**. is opened.
AT3.1.1. Instead of the default values mentioned in **AT2.1.1**. and **AT2.1.2.**, use the values of the opened association tag as defaults.
AT3.1.2. Clicking the „OK" button will store the changes made to the association tag, close the editing window and show the display window.

6.2 Model Transformation from SecureUML to UMLsec

The transformation requirements are followed in the order presented in this chapter. The requirements from SU1 to SU3 are consistent with the transformation rules presented in Chapter 3.3.1 and a summary of the actions performed within this transformation can be seen on Figure 9. First, after the user has clicked the respective menu element, an activity diagram is created. Then, activity partitions of resources and roles are added, along with respective actions. Finally, the user is presented with partial information about association tags that can be created, which they can join into complete tags. After the transformation, a note is created on the diagram, reminding the user to add information to complete the UMLsec model.

SU0: Initialisation

The requirements for the initial transformation activities are as follows:

SU0.1. The transformation can be activated upon clicking „SecureUML to UMLsec transformation" in the diagram context menu.
SU0.2. Check whether the diagram being transformed is a class diagram. If not, display an error message to the user and cancel transformation. If yes, continue.
SU0.3. Create a new activity diagram.
SU0.4. If the stereotype <<rbac>> has not been created, then created it. Assign the activity diagram the stereotype <<rbac>>.

SU1: Resources

The requirements for processing resource elements are as follows:

SU1.1. From the class diagram, capture the names of the classes with the stereotype <<secuml.resource>>. Create an activity partition in the activity diagram for each class name captured from the class diagram.
SU1.2. From each class with the stereotype <<secuml.resource>>, capture the list of its operations. For each operation captured, create an action with the same name.
SU1.3. Assign the operations created in **SU1.2.** to the activity partitions created in

SU1.1. An action originating from an operation should be assigned to the activity partition that originates from the class that owns the operation.

SU1.4. For each action created, capture a partial association tag {protected = [name of the action]}.

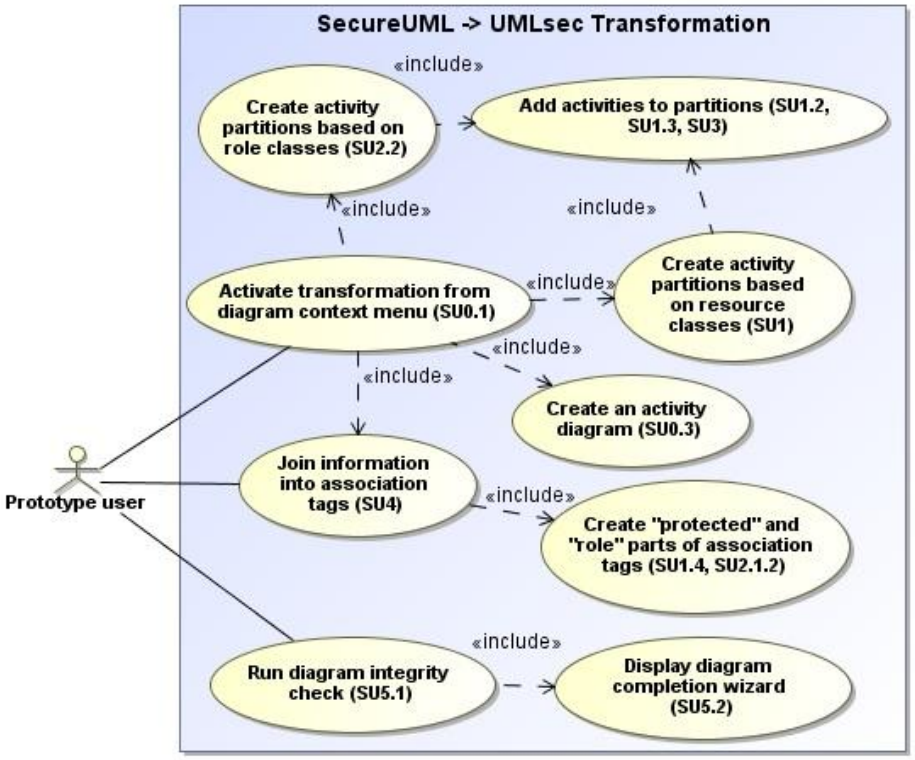

Figure 9. Use case diagram of SecureUML -> UMLsec transformation

SU2: Users and roles

The requirements for processing user and role elements are as follows:

SU2.1. For each relationship with the stereotype <<securml.assignment>>, do the following:

SU2.1.1. Check if the relationship connects a class with the stereotype <<securml.user>> to a class with the stereotype <<securml.role>>. If not, display an error message to the user and move on the next relationship (not done). If yes, continue.

SU2.1.2. Capture the names of the classes with the stereotypes <<securml.user>> and <<securml.role>>. Capture a partial association tag {role = [user name], [role name]}.

SU2.2. For each class on the class diagram with the stereotype <<securml.role>>, capture its name. Create an activity partition on the activity diagram carrying the captured name.

SU3: Permissions

The requirements for processing permission elements are as follows. For each association class with the stereotype <<securml.permission>>, do the following:

SU3.1. Check if the association class connects a class with the stereotype <<securml.role>> to a class with the stereotype <<securml.resource>>. If not, inform the user and move on the next association class (not done). If yes, continue.

SU3.2. Capture the names of the attributes of the association class. For each name captured, create an action carrying the same name on the activity diagram. Assign the action to the activity partition, which originates from the class with the stereotype <<securml.role>> that is related to the association class.

SU4: Association tags

The requirements for adding association tags to the UMLsec model are as follows:

SU4.1. Display a dialog window, which presents the user with partial association tags captured in **SU1.4.** and **SU2.1.2**. Enable the user to tick one {protected} tag and one {role} tag.

SU4.2. Display a „Join" button. When clicked, a full association tag is created:
{protected = [resource name]}
{role = [user name], [role name]}
{right = [role name], [resource name]}

SU4.3. Completion checks should be implemented – no two identical tags can be created and no tags with any missing information can be created.

SU4.4. Display the all the full tags below. Also display an „Unjoin" button next to each association tag, which removes the tag from the list.

SU4.5. Store the created association tags in the UMLsec model.

SU5: Diagram completion wizard

The requirements for running a SecureUML completion check are as follows:

SU5.1. Run a completion check on the created activity diagram. The check must also be independently available from the diagram context menu. Hence, first it needs to be checked, whether the diagram is an activity diagram and whether it carries the stereotype <<rbac>>. If not, display an error message to the user and cancel the completion check. If yes, continue.

SU5.2. Create a note on the diagram reminding the user to add missing elements. Display the following information:

SU5.2.1. If no initial node is present, then a reminder to create one;

SU5.2.2. If no final node is present, then a reminder to create one;

SU5.2.3. Names of the actions that are not associated with any control flows.

6.3 Model Transformation from UMLsec to SecureUML

The following actions are automatically carried out after initiating a transformation, in the order they are presented in this chapter. The requirements from US1 to US3 are consistent with the transformation rules presented in Chapter 3.3.2 and a summary of the actions performed within this transformation can be seen on Figure 10. First, a class diagram is created. Then, classes carrying information on resources and roles are added, based on activity partitions and information in association tags. Following that, users are extracted from association tags and also added to the diagram as classes. Association classes containing permissions are added last. After the transformation, a note is created on the diagram, reminding the user to add the necessary information to complete the SecureUML model.

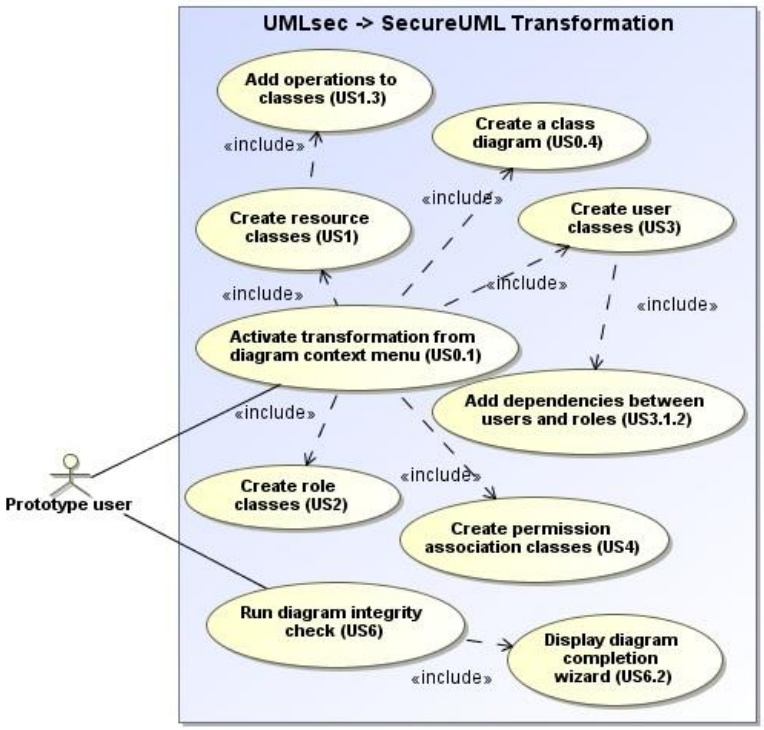

Figure 10. Use case diagram of UMLsec -> SecureUML transformation

US0: Initialisation

The requirements for the initial transformation activities are as follows:

US0.1. The transformation can be activated upon clicking „UMLsec to SecureUML transformation" in the diagram context menu.

US0.2. Check whether the diagram being transformed is an activity diagram. If not, display an error message to the user and cancel transformation. If yes, continue.

US0.3. Check whether the activity diagram is of the stereotype <<rbac>>. If not, display an error message to the user and cancel transformation. If yes, continue.

US0.4. Create a new class diagram.

US1: Resources

The requirements for processing resource elements are as follows:

US1.1. From the activity diagram, capture all the names of the actions that are defined by the association tag {protected = <action>}.

US1.2. For each action captured, find the respective activity partition it belongs to, and in the class diagram create a class with the name of that activity partition and the stereotype <<secuml.resource>> (unless it has already been created).

US1.3. Add all actions captured in **US1.1** to the respective classes created in **US1.2** as operations.

US2: Roles

The requirements for processing role elements are as follows:

US2.1. For each activity partition that was NOT captured in **US1.2**, create a class in the class diagram and add the stereotype <<secuml.role>> (unless it has already been created).

US3: Users

The requirements for processing user elements are as follows:

US3.1. For each association tag {role = <user>, <role>}:

US3.1.1. Create a class with the name <user> and stereotype <<secuml.user>> (unless it has already been created).

US3.1.2. Create a dependency link with the stereotype <<assignment>> between the class created in **US3.1.1.** and the <<secuml.role>> class with the name <role> captured in **US2.1**.

US4: Permissions

The requirements for processing permissions are as follows:

US4.1. For each occurrence of the association tag {right = <role>, <action>}, create an association class between the <<secuml.resource>> which holds the operation with the name <action> and <<secuml.role>> corresponding to <role>. Assign the stereotype <<secuml.permission>> to it (not done).

US4.2. Find actions that happen directly before <action> in the activity diagram and that belong to the activity partition with the name <role>. Add those actions as operations to the association class between the respective <<secuml.role>> and <<secuml.resource>>.

US6: Diagram completion wizard

The requirements for running a UMLsec completion check are as follows:

US6.1. Run a completion check on the created class diagram. The check must also be independently available from the diagram context menu. Hence, first it needs to be checked, whether the diagram is a class diagram. If not, display an error message to the user and cancel the completion check. If yes, continue.

US6.2. Create a note on the diagram reminding the user to add missing elements. Display the following information:

- **US6.2.1.** Names of the classes with the stereotype <<secuml.role>> which have no attributes;
- **US6.2.2.** Reminder to add authorisation constraints;
- **US6.2.3.** Names of the classes between which lies a relationship that has no multiplicities defined;
- **US6.2.4.** Names of the classes between which lies a relationship that has no end names defined;
- **US6.2.5.** Action types for all the identified actions (not done).

6.4 Design

For an illustration of the main details of the plug-in implementation, see Figure 11 (the diagram does not depict completion checks). To create a plug-in for MagicDraw UML, it is necessary to create a class that extends the MagicDraw API class *Plugin*. In the case of this prototype, that class is *TransformationPlugin*. The *init()* method implemented in the class extending *Plugin* describes the actions done when the plug-in is loaded. In this case, elements in the context menu are created: transformations, completion checks and association tags view. A menu action is a class extending the MagicDraw API class DefaultDiagramAction. The name of the menu element is defined in the DefaultDiagramAction constructor and its functionality is described in the *actionPerformed()* method.

The diagram action classes (*SU_Transformation* and *US_Transformation*) contain the transformation functionality. Firstly, elements of the initial diagram are read. The API includes classes for all sorts of entities to enable that – for a project, different diagrams, diagram elements, and their properties. Then, the information collected is processed, and a new diagram is created. For each project, a factory can be queried in the API – an instance of the class *ElementsFactory* – through which it is possible to create new diagrams and their elements. Therefore, a large part of the transformations' functionality uses the factory.

Both transformations also make use of UMLsec association tags. To make the transformations structured and enable the user to work with the tags, a new class was created - *AssociationTag*. Information on the SecureUML diagram is used to create new association tags in the process of transformation (requirements **SU1.4.** and **SU2.1.2**), and UMLsec transformation uses them to create SecureUML resource, role and user classes (requirements **US1.1**, **US3.1** and **US4.1**). Support for viewing, adding and editing tags in a UMLsec model has also been developed, which lies in the *AssociationTagWindow*

class. It is a component which features many visual elements and much of its functionality resides in reacting to clicking different buttons or choosing menu elements – such listeners are stored in the *buttonlisteners* package. The respective context menu element is created within the *ShowAssociationTags* class.

Detailed and more complete class diagrams of the prototype's code, along with explanations, can be found in Chapter 7.

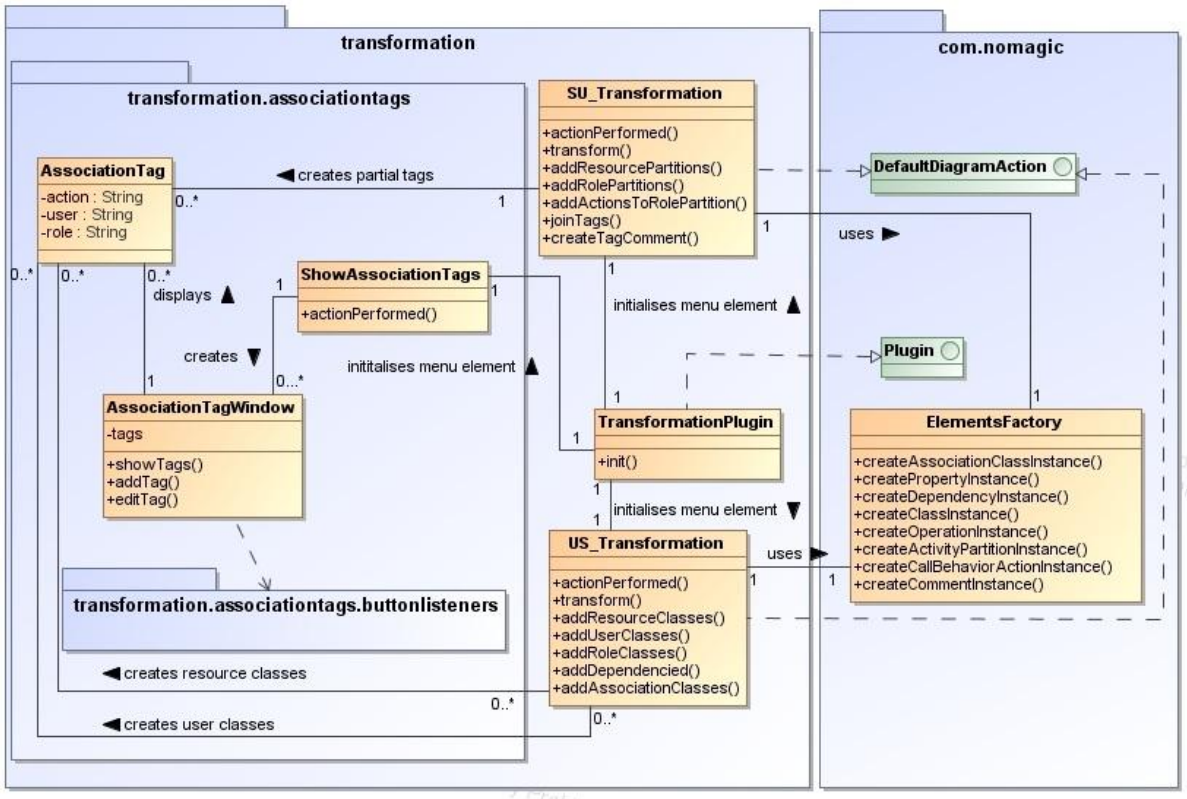

Figure 11. Class diagram of the prototype

6.5 Summary

The requirements, as well as the design of the prototype tool, are divided into three parts: two transformations and support for editing UMLsec association tags. Both transformations go through step-by-step actions to collect information from one model and carry it to the other. The transformations finish off by adding a note on the transformed diagram stating which elements are still missing from a complete model. This completion check can be initiated at any time from the diagram context menu. On a UMLsec model, all its association tags can be viewed and edited.

The implementation requires extending MagicDraw API classes, in order to create both a plug-in and new elements in the diagram context menu. For the creation of new elements on the transformed diagram, instances of the API class *ElementsFactory* are used.

Chapter 7: Implementation

In this chapter, the prototype's code structure is described. It is illustrated with detailed class diagrams, which include each class's attributes and methods, and dependencies between different classes and packages. The MagicDraw API classes are not represented. The classes written for transformations and completion checks can be seen on Figure 13. Figure 14 goes into the details of association tags creation in SecureUML to UMLsec transformation. Figure 15 gives the structure of the association tags component's general elements. The functionality that enables manipulating association tags is implemented in different buttons, which are described in Figure 16.

In Figure 12, the prototype's package structure is presented. The code is separated into four separate sections that interact with each other: two transformations, association tags and diagram completion checks (wizards).

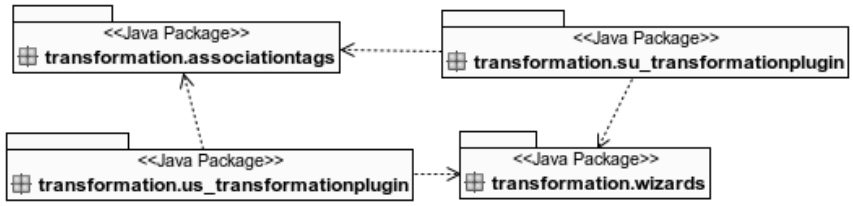

Figure 12. Package structure of prototype code

Figure 13 presents the class diagram of the transformations and wizards. The centre class of the plug-in is *TransformationPlugin*, which extends the API class *Plugin* and initialises all the menu elements (Figure 17 shows how they appear in the diagram context menu). Both transformation classes (*SU_Transformation* and *US_Transformation*) extend the API class *DefaultDiagramAction*, which means that they appear as diagram context menu elements. Clicking the elements triggers the transformation functionality, which lies in the implemented method *actionPerformed()*. The transformations feature separate methods for different sections of the requirements (described in Chapters 6.2 and 6.3). Both transformations call corresponding wizards when they are finished. However, both wizards can also be called from the diagram context menu to run checks on SecureUML or UMLsec diagrams that the user has created or improved on their own. For that purpose, two more classes extending the *DefaultDiagramAction* class exist – *SecureUML_WizardDiagramAction* and *UMLsec_WizardDiagramAction*. Since information on association tags is stored on a non-visible comment on the UMLsec diagram, both transformations have a respective method for accessing it. *SU_Transformation* features the *createTagComment()* method, which creates a comment in the UMLsec model, containing data on the association tags captured from the SecureUML model. *US_Transformation*, in turn, features the *extractTags()* method, which finds the comment containing association tags and creates instances of the *AssociationTag* class out of the information found.

Figure 14 gives a more detailed picture regarding SecureUML -> UMLsec transformation and the classes handling the process of joining association tags, which

requires user input. It is implemented in the form of a pop-up window and the functionality lies in classes implementing the Java *ActionListener* interface, which are attached to different buttons on the pop-up window.

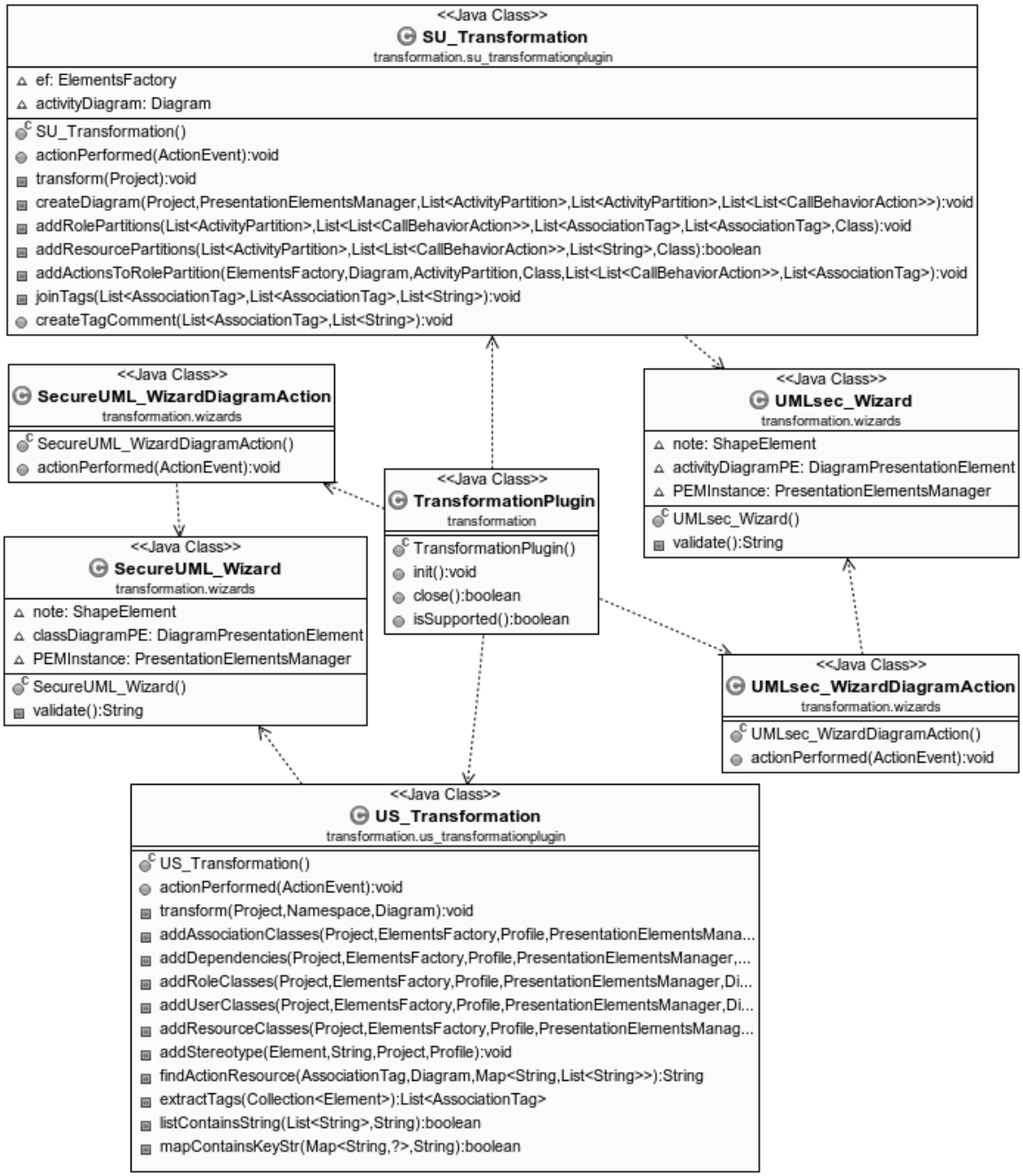

Figure 13. Transformations and diagram completion wizards.

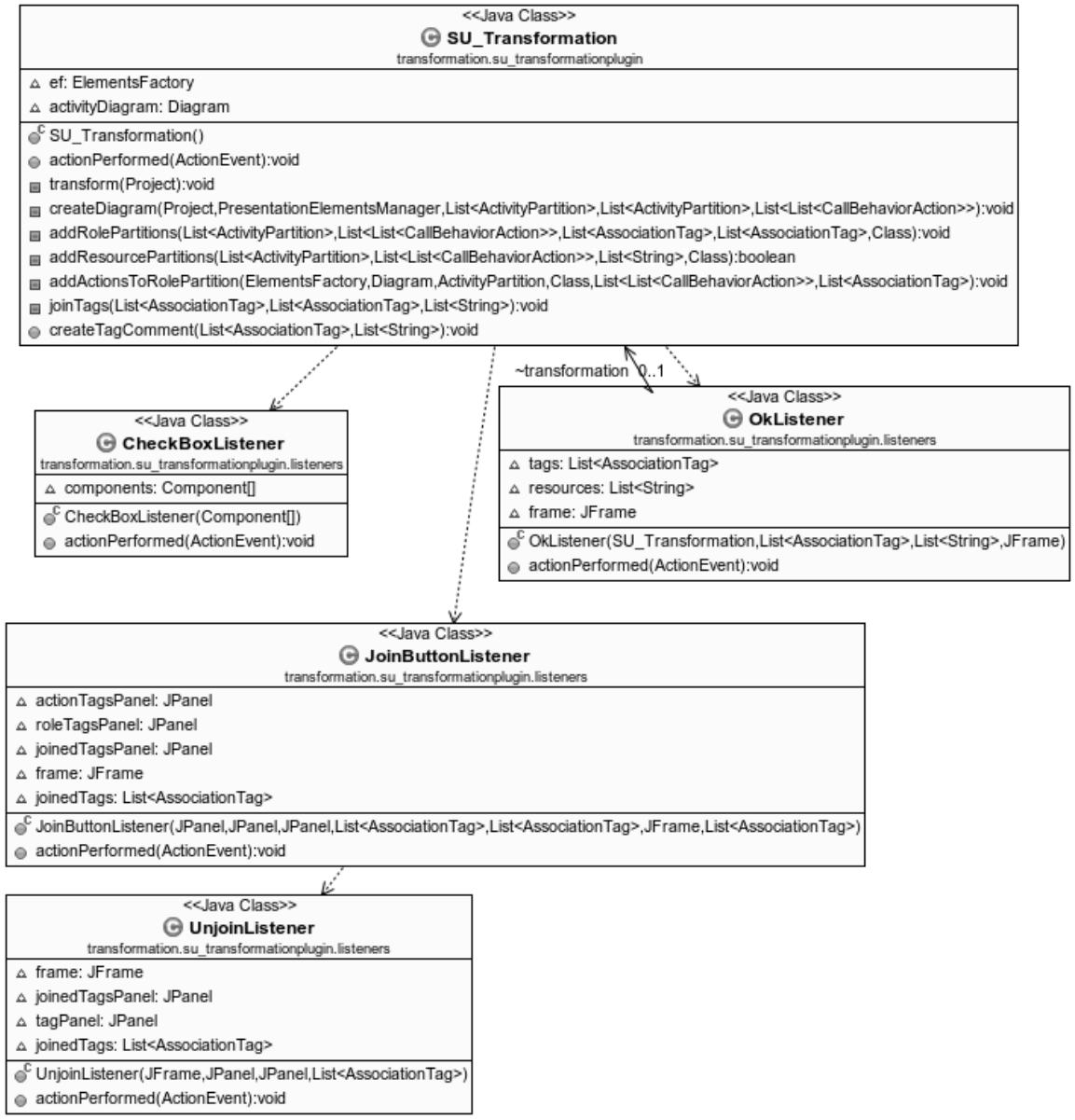

Figure 14. SecureUML to UMLsec transformation (with joining association tags)

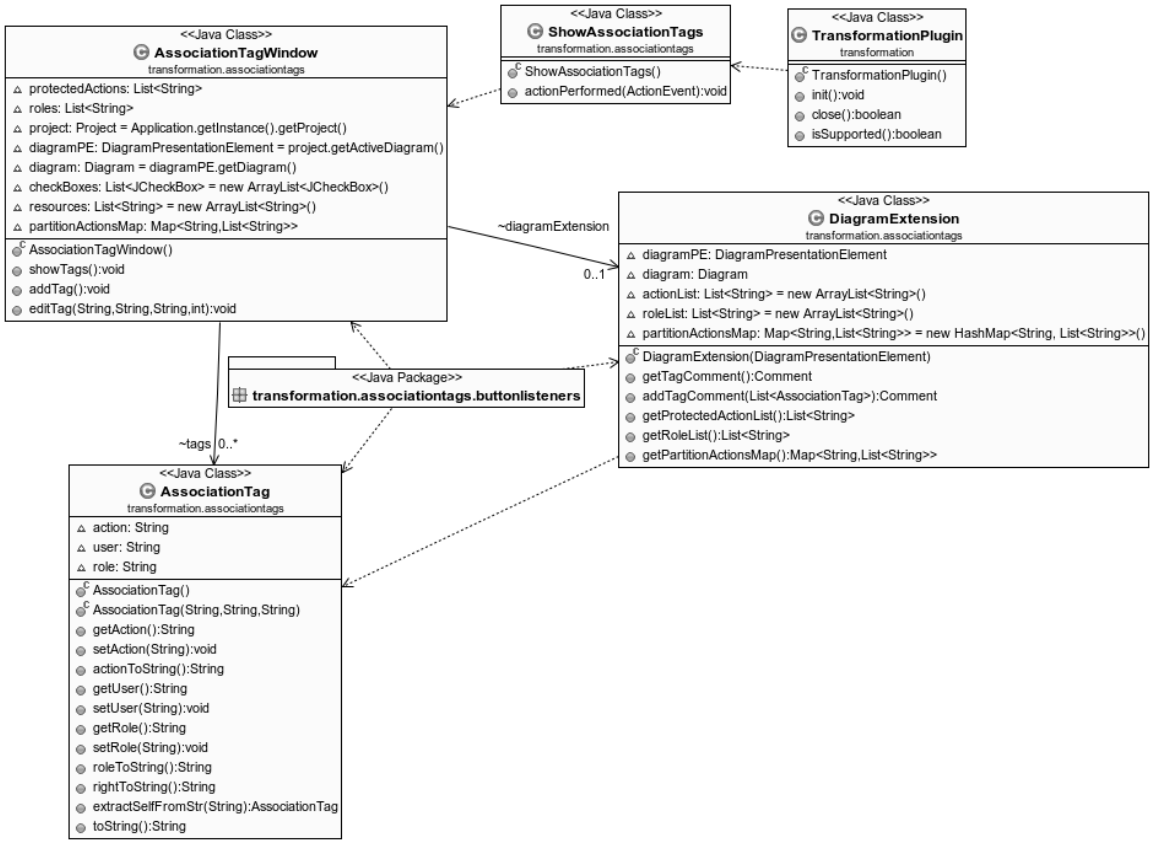

Figure 15. Association tags

On Figure 15, the structure of the *associationtags* package can be seen. Instances of the *AssociationTag* class exist virtually and only while the transformation is ongoing or the association tags window is open. At other times, information on a UMLsec model's association tags is stored in a non-visible comment. *ShowAssociationTags* represents the respective menu item by extending the *DefaultDiagramAction* API class and is also initialised by *TransformationPlugin*. It calls the *AssociationTagWindow* class, which displays a pop-up window and has methods for showing, adding and editing association tags. The functionality, again, lies in classes implementing the Java *ActionListener* class, which are collected in the *associationtags.buttonlisteners* package. Those classes can be seen in detail on Figure 16. *AssociationTagWindow* also uses the class *DiagramExtension* in order to read information on association tags from the respective comment in the UMLsec model.

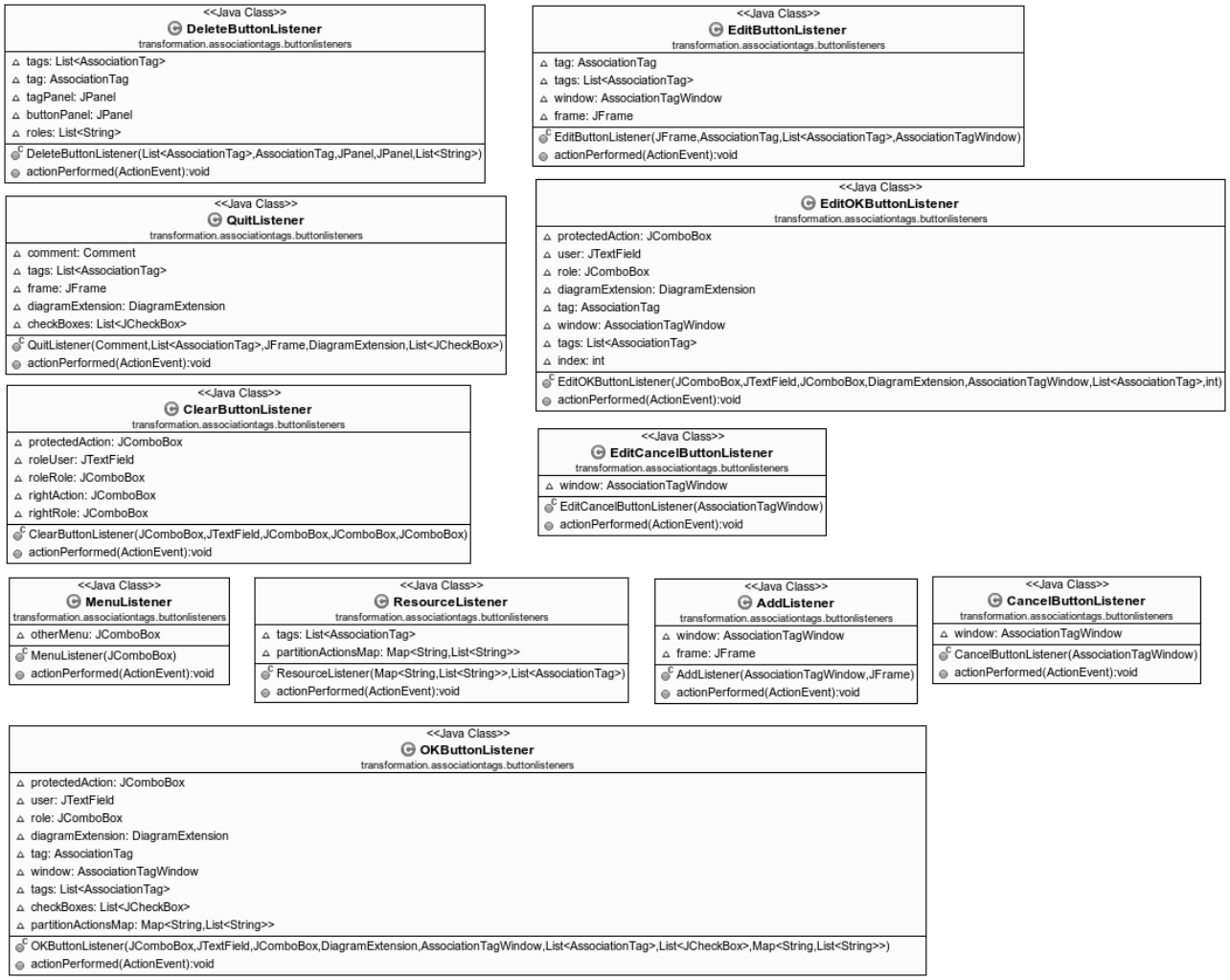

Figure 16. *ActionListeners* for the association tags window

Chapter 8: User Interface Manual

The following user manual will pose as a graphical guide through using the transformation tool and its different features. It is expected that the user of the transformation tool already has knowledge on how to create role-based access control models in the SecureUML and UMLsec modelling languages. Sample *Meeting Scheduler* [6] models in both languages can be found in Appendix D.

1. To install the plug-in, copy the *transformation* folder from Appendix C to the *MagicDraw/plugins* directory and run the application.
2. Open the SecureUML or UMLsec diagram you wish to transform.
3. Make a right-click (secondary click) on the diagram. All the activities of the prototype can be activated from the diagram context menu (see Figure 17)

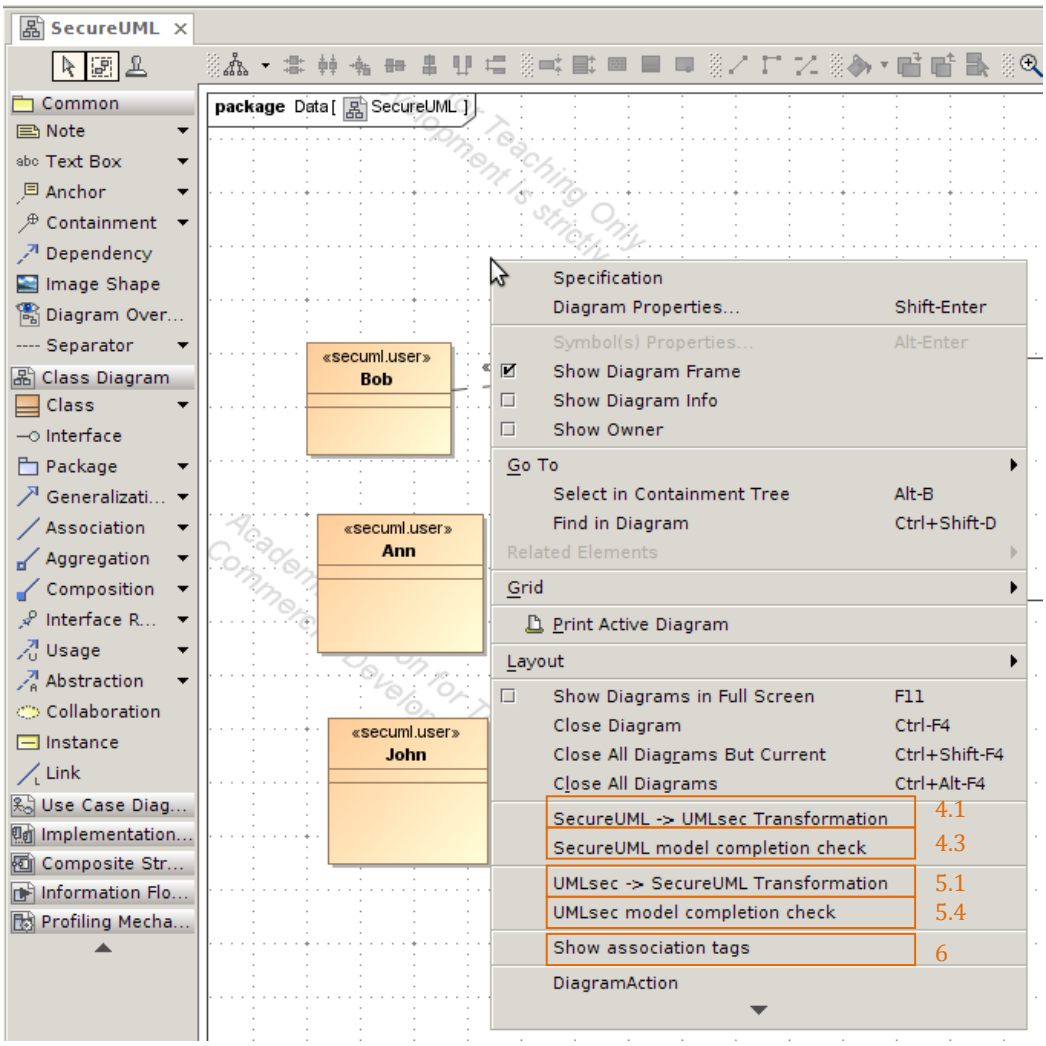

Figure 17. Diagram context menu

4. UMLsec to SecureUML transformation
    4.1. If you wish to transform a UMLsec diagram, click "UMLsec -> SecureUML Transformation" (see Figure 17).
    4.2. The model is transformed. See the wizard in the corner of the diagram (Figure 18) – it states, which elements are still missing from a complete SecureUML diagram.

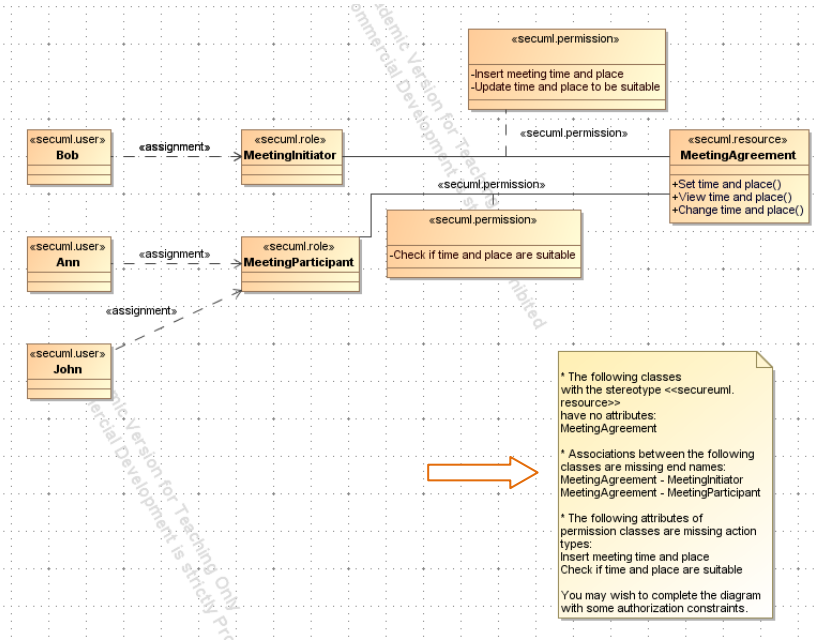

Figure 18. SecureUML completion wizard

    4.3. If you have added some missing elements and wish to run the wizard again, choose "SecureUML model completion check" from the diagram context menu (see Figure 17).
5. SecureUML to UMLsec transformation
    5.1. If you wish to transform a SecureUML diagram, click "SecureUML -> UMLsec Transformation" in the diagram context menu (see Figure 17).
    5.2. The tool collects information from the diagram that can be used as a basis for creating UMLsec association tags. However, the information is not complete – names of protected actions and user-role relationships are collected, but they need to be manually joined together. For that purpose, they are displayed in a pop-up window (see Figure 19). To join tags, tick one action and one user-role relationship and click "Join". "Unjoin" will remove the created association tag.

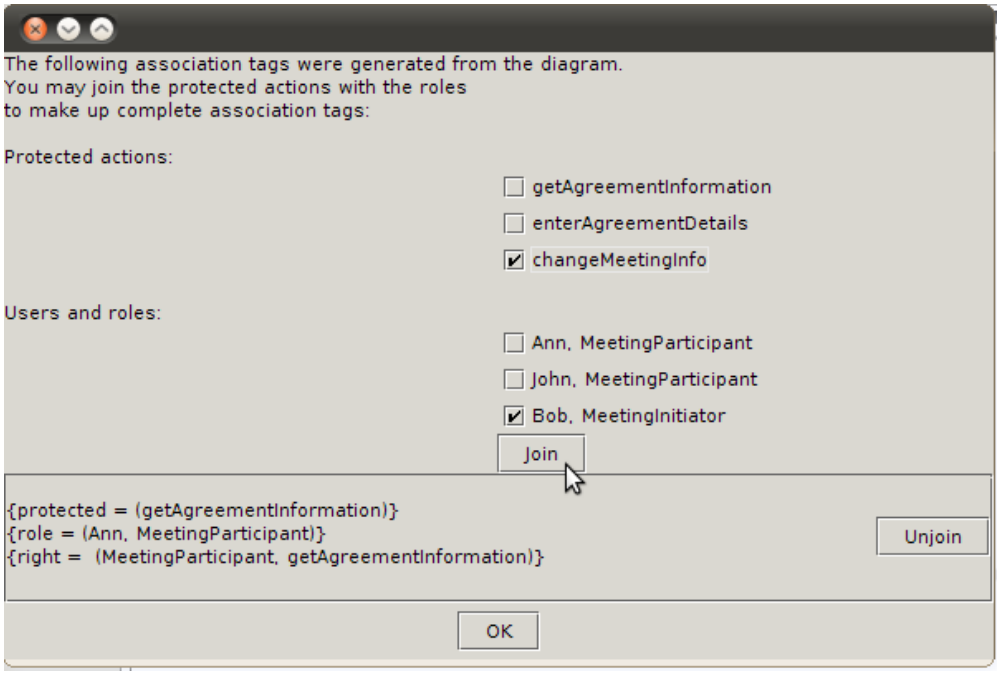

Figure 19. Joining association tags

5.3. After clicking "OK" (see Figure 19) the model is transformed. See the wizard in the corner of the diagram – it states, which elements are still missing from a complete UMLsec diagram (see Figure 20).

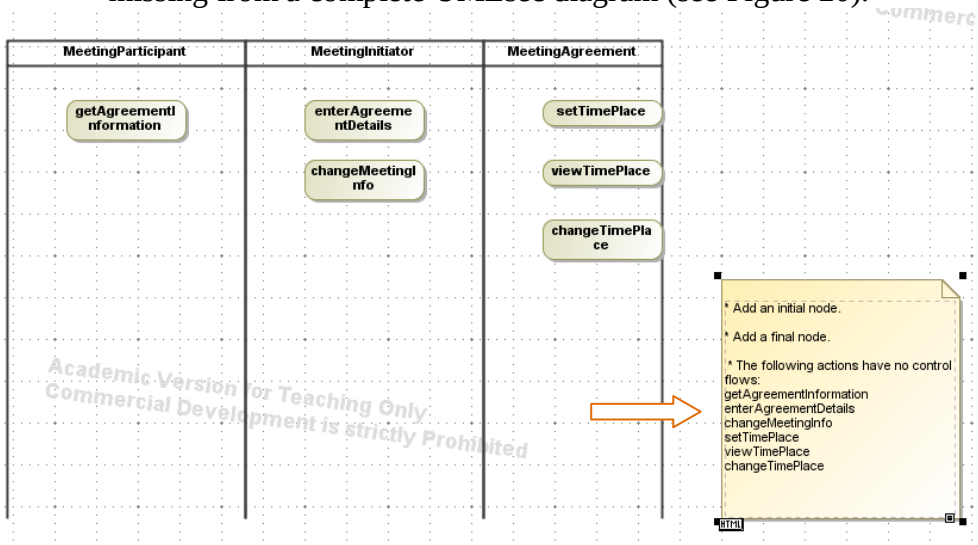

Figure 20. UMLsec completion wizard

5.4. If you have added some missing elements and wish to run the wizard again, choose "UMLsec model completion check" from the diagram context menu (see Figure 17).

6. To view association tags, edit them or add new ones, choose "Show association tags" from the UMLsec diagram context menu (see Figure 17). Doing so will open a pop-up window that displays all the association tags of the model (see Figure 21).

6.1. In the uppermost panel (see Figure 21), tick the names of the activity partitions, which correspond to protected resources.

Figure 21. Association tags

6.2. "Add" (see Figure 21) will open a dialog window (see Figure 22), where you can create a new association tag.

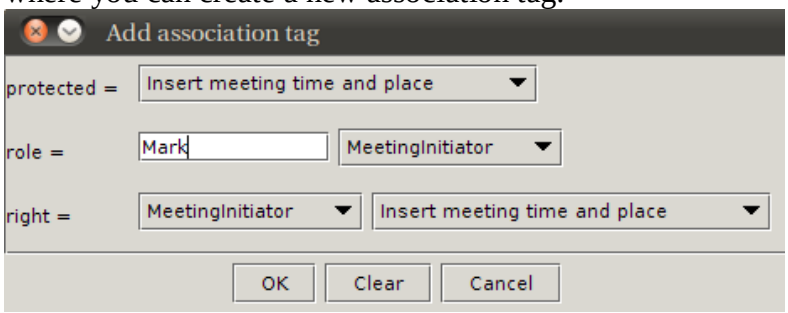

Figure 22. Adding an association tag

6.3. "Edit" (see Figure 21) will open a similar dialog window to Figure 22, pre-filled.

6.4. "Delete" (see Figure 21) will remove the association tag from the list.

Chapter 9: Conclusion

The aim of this thesis is to develop a prototype tool for security model transformations. For the purpose of creating this prototype, NoMagic Inc. has given permission to use the OpenAPI of its professional modelling tool MagicDraw UML. The application's level of detail and the possibilities provided by the API enable free low-level manipulation of models.

Role-based access control is a widespread method for implementing a computer system's security, by creating different roles – fixed sets of privileges to perform actions in the system – and assigning them to users with respective job duties. Designated UML standards exist to aid the design of role-based access control. One of them is SecureUML, an extension of the UML class diagram. The second, UMLsec, uses UML activity diagrams.

The main product of this thesis is a tool developed for performing transformations from the SecureUML language to the UMLsec language and *vice versa*, with the additional features of model completion wizards for both languages and support for viewing, editing and adding UMLsec association tags. Documentation is another product of the thesis, consisting of requirements analysis, design descriptions, code documentation and a user manual. The requirements have been compiled in the form of step-by-step functionality-related instructions. Secondly, the design of the prototype's code has been explained with the aid of class diagrams, showing the interaction of the code's classes with the MagicDraw API and each other. Thirdly, code documentation is provided in the Javadoc format. Finally, a user interface manual has been created, which goes through the main use cases of the prototype tool.

The tool's purpose is to demonstrate the possibility to automate creating multiple coherent security models. The transformed model's consistency with the initial model is assured by implementing transformation rules, which are the result of studies [4] and [5]. Also, since a considerable amount of the new model is created automatically, it can be expected that it will take less time to complete it, than to manually compile the whole of the new model. This, however, is a subjective evaluation, since validation has not been performed to the tool within this thesis.

There are a few limitations to the prototype, which can be bettered within future development. For one, some requirements were not implemented in this version of the transformation tool. The code could also be optimised and refactored to meet the requirements of good programming style. It is possible that the MagicDraw API enables developing whole new model types for SecureUML and UMLsec, which might address the problem of lack of support for authorisation constraints and association tags better than the current solution. Furthermore, a validation should be performed to the prototype. Following that, improvements with regard to usability and usefulness could be made to the requirements analysis of the transformation tool and the tool itself. A more significant future possibility is studying how the prototype works with other models than the Meeting Scheduler example. It may be that problems arise, when tested with more complex diagrams, which need further analysis of the transformation rules or requirements for the tool.

Rollipõhise juurdepääsukontrolli mudelite teisendamise prototüüp

Rollipõhine juurdepääsukontroll on arvutisüsteemides laialtkasutatav mehhanism – see tagab turvalisuse, lubades ligipääsu ressurssidele vaid nendele kasutajatele, kel on selleks vastavad õigused. Rollipõhise juurdepääsukontrolli lahendusi on võimalik välja töötada selliste modelleerimiskeelte abil, nagu SecureUML ning UMLsec, mis mõlemad esitavad süsteemi disaini erinevatest vaatepunktidest. Mitme kooskõlalise mudeli koostamine võib aga osutuda keeruliseks ning aeganõudvaks ülesandeks. See võib omakorda vähendada rollipõhise juurdepääsukontrolli mudelite loomise motivatsiooni.

Ühe lahendusena võib pakkuda arendajale tööriista, mis kasutaks ühes keeles loodud mudelit, et selle põhjal automaatselt konstrueerida mudel teises keeles. Teisendatud mudel aga ei oleks täielik, kuna eelmainitud keeli kasutatakse osalt erineva informatsiooni kandmiseks. Tööriista eesmärk oleks vähendada vajadust teist mudelit koostades käsitsi informatsiooni kopeerida.

Selle töö raames arendatakse tööriista prototüüp, mis teisendab SecureUML mudeli UMLsec mudeliks ning vastupidi. See teostatakse Java programmeerimiskeeles ning pistikprogrammina professionaalsele UML modelleerimistööriistale MagicDraw. Rakendusele lisatakse menüüpunktid, millele vajutades käivitatakse teisendused: SecureUML keelest UMLsec keelde või vastupidi. Lisafunktsioonina arendatakse ka mõlema mudeli täielikkuse kontrollid, mille abil antakse kasutajale teada, kas kõik vajalikud elemendid on olemas. Need annavad kasutajale juhtnööre, kuidas teisendatud mudelit täiendada, kuna on teada, et pärast teisendust on teatud info uuelt mudelilt puudu. Teine lisakomponent võimaldab töödelda UMLsec märgendeid (ingl. k. *association tags*), mis on SecureUML ning UMLsec vaheliste teisenduste tähtis osa. Käesoleva töö raames on koostatud ka pistikprogrammi dokumentatsioon – nõuete analüüs, koodi dokumentatsioon ning kasutusjuhend – mille eesmärk on tagada prototüübi mõistmine ning aidata kaasa selle edasiarendamisele tulevikus.

Bibliography

1. Sandhu, R.S., Coyne, E.J.: Role-Based Access Control Models (1996). In: *IEEE Computer*, Volume 29, Number 2, pp 38-47, IEEE Press.

2. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control (2001). In: *ACM Transactions on Information and System Security*, Volume 4, Number 3, pp 224-274, ACM New York.

3. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructures (2005). In: *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 15, Issue 1, pp 39-91, ACM New York.

4. Matulevičius, R. and Dumas, M.: A Comparison of SecureUML and UMLsec for Role-Based Access Control (2010). In: *Proceedings of the 9th International Baltic Conference (Baltic DB&IS 2010)*, University of Latvia Press, pp 171-185.

5. Matulevičius, R. and Dumas, M.: Towards Model Transformation between SecureUML and UMLsec for Role-Based Access Control (2011). In: *Proceedings of the 2011 conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, DB&IS 2010*, IOS Press Amsterdam, pp 339-352.

6. Feather, M. S., Fickas, S., Finkelstein, A., van Lamsweerde, A.: Requirements and Specification Exemplars (1997). In: *Automated Software Engineering*, 4 (4), pp 419–438.