Verifying Security Requirements using Model Checking Technique for UML-Based Requirements Specification

Yoshitaka Aoki
Graduate School of Engineering and Science,
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku,
Saitama-City, Saitama 337-8570, Japan
matsuura@se.shibaura-it.ac.jp

Saeko Matsuura
Graduate School of Engineering and Science,
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku,
Saitama-City, Saitama 337-8570, Japan
matsuura@se.shibaura-it.ac.jp

***Abstract*—Use case analysis is known to be an effective method to clarify functional requirements. Security requirements such as access or information control tend to increase the complexity of functional requirements, and therefore, need to be correctly implemented to minimize risks. However, general developers find it difficult to correctly specify adequate security requirements during the initial phases of the software development process.**

We propose a method to verify security requirements whose specifications are based on Unified Modeling Language (UML) using the model checking technique and Common Criteria security knowledge. Common Criteria assists in defining adequate security requirements in the form of a table. This helps developers verify whether UML-based requirements analysis models meet those requirements in the early stages of software development. The UML model and the table are transformed into a finite automaton in the UPPAAL model checking tool.

***Index Terms*—UML, Security Requirements, Verification, Model Checking, Common Criteria, Access Control**

I. Introduction

In this study, we primarily focus on access control to services, among all security requirements, because it is strongly related to the components of functional requirements and it often tends to increase the complexity of the entire set of requirements. However, to avoid both errors and omissions of security requirements for target services, application developers are required to have in-depth knowledge of the related security functions.

The Common Criteria for Information Technology Security Evaluation (Common Criteria or CC) [1] is an international standard (ISO/IEC 15408) for computer security certification. The CC consists of three parts. Part 1 includes the general model and the important terminology. Part 2 is a catalogue of security functional components that are the basis for the security functional requirements expressed in a Protection Profile (PP) or a Security Target (ST) and are implementation-dependent security needs for the target system. Security functional requirements in CC are expressed as classes, families, and components. Each functional component provides a complete list of its dependencies on other security functional components. This helps developers to check whether all necessary security requirements have been extracted. Moreover, security functional requirements are usually defined by multiple Security Function Policies (SFPs) that represent the rules the target system must enforce. Each such SFP must specify its scope of control, by defining the subjects, objects, resources or information, and operations to which it applies.

Use case analysis [2] is known as an effective method to clarify functional requirements. We propose a method for model-driven requirements analysis [3, 4, 5] using the Unified Modeling Language (UML) [6]. This method defines a use case with an activity diagram that consists of several action flows, including object nodes with a class classifier. Access control is strongly related to components such as actions and object nodes in a use case; therefore, the increased branches often make the entire set of requirements complicated. We will demonstrate that our method can correctly define adequate security requirements that enhance requirements specification quality in the early stages of software development.

Figure 1 presents an overview of the proposed method to verify SFPs for UML requirements analysis models (RA Model) with the model checking technique.

First, security requirements are defined as SFPs by relating them to a UML-based requirements specification according to the CC guide. Second, the use cases are translated into a system model, so that a model checking tool can verify their adequacy. This can be done because use cases are behavioral models and SFPs define logical expressions between attributes of classes.

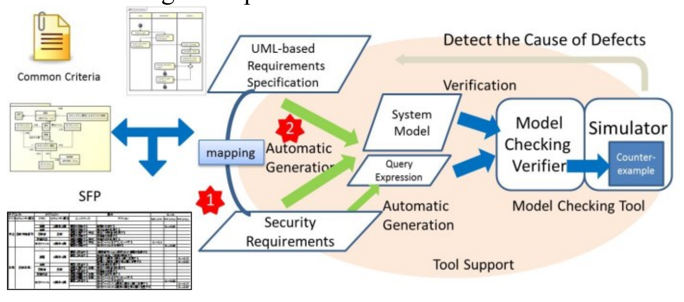


Fig. 1. Overview of Verification Method

The rest of this paper is organized as follows. Section II describes security requirements verification problems in the early phases of software development from CC and model checking perspectives. Section III explains how to define a UML requirements analysis model. Section IV explains how to define SFPs against the UML model. Section V explains our verification method using model checking. Finally, section V discusses our results, conclusions and future research directions.

II. Security Requirements Verification Problems

A. UML Model Security Requirements Definition

Application developers generally have minimal security knowledge. CC, on the other hand, has the potential to serve as a structured guideline for developers to describe security requirements in terms of appropriate functional components for a target system. Therefore, a developer can use the CC as a catalog for security knowledge. Each component in the CC is expressed by the terms in Part 1.

However, the descriptions are highly abstract, because they need to be applicable to different types of information systems.

We need to solve the following problems to enable application developers to use the CC as a source of supplementary security knowledge. We assume target system functional requirements are defined with the RA model. Threats against the target system are analyzed based on all the data in the RA model to protect RA model data against malicious users. First, we select some adequate components that are countermeasures to the threats from the catalog and then we define SFPs for the selected components. Second, we combine these SFPs with both the class diagram (data) and the activity diagram (actions) in the RA model, because the divided security requirements should be correctly combined with the functional requirements.

We define a mapping rule between elements in the RA model and the SFPs, because all SFPs have to be implemented through application to the target system.

B. Model Checking Technique Application Problems

The Model checking technique is regarded as an effective method to improve reliability in the early stages of software development. The model checking tool uses temporal logic to model the system as a network of automata extended with integer variables, structured data types, user defined functions, and channel synchronization. Based on these properties a system model and query expressions can be defined that specify which properties are to be checked. When the specified properties are not satisfied, the tool provides counterexamples that show how the properties can be falsified. The simulator helps to detect defect causes by tracing the processes in which the counterexamples occur.

Model checking technique automatically verifies a model, by exhaustively checking all paths to detect properties that developers often overlook. However, developers typically find it difficult to define an appropriate model and formulas at all times, because the path and state formulas should be defined by items in the model.

Path formulas can define properties such as reachability, safety, and liveness. Reachability means that the specified state will be reached at some point in time. Safety means that something bad will never happen. Liveness means that something expected will eventually happen. State formulas need to be defined by expressions related to several process IDs or variables of the state.

Our RA model defines use cases with an activity diagram that includes several user and system action sequences that represent both normal and exceptional use case flows. Access control is added by defining an action with the guard conditions expressed by logical expressions of the security attributes added to the entity classes in the RA model.

Model checking can automatically translate these logical expressions into query expressions and then the specified safety properties can be verified against the RA model.

III. UML based Requirements Analysis Process

In this section, we explain our method using the requirements of an existing system as an example.

A. Example Requirements

LUMINOUS [7] is a learning management system in our department that enables teachers and students to manage learning materials, reports, questionnaires, etc. We apply our security requirements analysis method to the development of a new Bulletin Board System (BBS) in LUMINOUS. There are two kinds of actors, teacher and student. The BBS has three use cases; a student can post a question, a teacher can answer a question, and both can view topics. Both types of actors can attach a file to the question or answer if necessary and can download an attached file while viewing a topic. A teacher can post public anonymous questions and answers if necessary. Then students can read only public questions and answers.

The Security requirement for the LUMINOUS BBS is protection of personal topics, including attached files, without being contrary to the teacher's intentions.

B. Functional Requirements Definition

We proposed a method of model-driven requirements analysis using UML. It is important to clearly model the interactions in use case analysis, because what customers want to do obviously appears in the interaction between a user and a system.

We specify a use case from the following four viewpoints to answer the associated questions:

- Application requirements: what kinds of input data and conditions are required to execute a use case as expected?
- Application requirements observation: what kinds of conditions should be required when not executing the use case? Moreover, how should the system treat these exceptional cases?
- Use case conditions: what kinds of behaviors are required to execute the use case?
- Output: What kinds of data outputs result from these behaviors?

Both process flow and entity data, which are required to execute the target application requirements, are defined with

UML activity diagrams and a class diagram based on the aforementioned four viewpoints and questions.

Activity diagrams specify not only normal and exceptional action flows, but also data flows that are related to these actions. Actions are defined by action nodes and data is defined by object nodes that are classified as members of a class that is defined in a class diagram. Accordingly, these two kinds of diagrams enable us to specify application process flows in connection with the data. This is one of the advantages of our method to use activity diagrams and class diagrams to specify security requirements. The interaction between a user and a system especially includes requisite flows and data on user input, conditions, and output to execute a use case correctly.

The second feature of our method is an activity diagram that has three types of partitions: user, interaction, and system. These partitions enable ready identification of the following activities: user input, interaction between a user and system caused by the conditions for executing a use case, and the resulting output. Object nodes in the user, interaction, and system partitions represent input data, output data, and entity data, respectively. The requirement analysis model is defined using a modeling tool astah*[8].

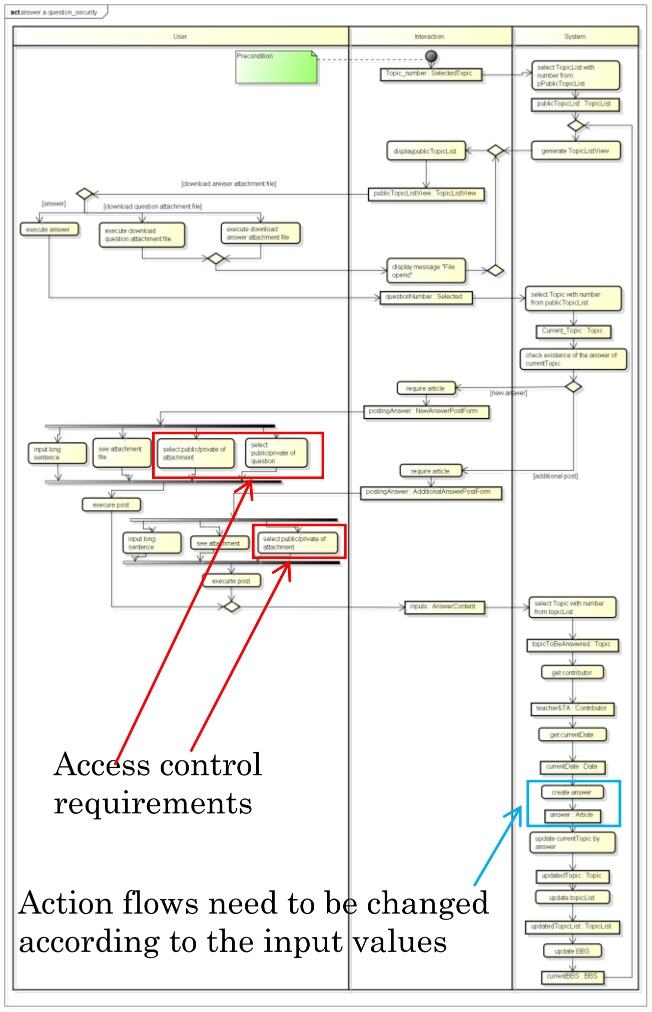

act answer a question_security
User
Interaction
System
Precondition
Access control requirements
Action flows need to be changed according to the input values

Fig. 2. A Use Case Model of Post a Question

IV. Security Requirements definition for UML Requirement Analysis Model using CC

A. *Misuse Case Threat Analysis*

It is conceivable that authenticated students could be malicious users, because LUMINOUS already has an authentication mechanism. All LUMINOUS BBS entity data are extracted from the RA model. The candidates for assets to be protected from malicious students are the entity data used in the only two use cases in which a student can be an actor. Attached files and topics may include information that the general population of public students should not read. This is a user data protection issue. Moreover, there is a privacy issue, because a topic includes the student's name. Therefore, attached files and topics themselves should be assets to be protected from malicious students. As a result, three misuse cases [9] were defined as threats conducted by malicious students (see Figure 3). Figure 2 shows an example of the use case "answer a question". When a teacher creates an answer he/she decides whether or not the topic is open to the students. Moreover, if there are files attached to the question and the answer, he/she also decides if they are open to the public.

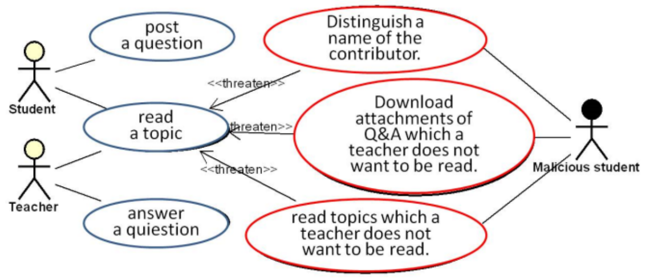

Student
Teacher
post a question
read a topic
answer a quiestion
<<threaten>>
<<threaten>>
<<threaten>>
Distinguish a name of the contributor.
Download attachments of Q&A which a teacher does not want to be read.
read topics which a teacher does not want to be read.
Malicious student

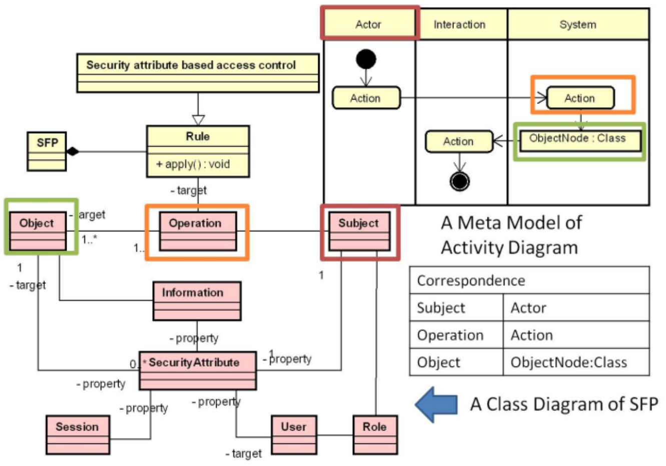
Fig. 3. Misuse Case for LUMINOUS BBS

B. *UML RA Model Security Requirements Mapping*

Based on the misuse case analysis we specify two kinds of threats, i.e., USER DATA PROTECTION and PRIVACY, using CC Part 2 as a catalog. The USER DATA PROTECTION class FDP contains families that specify user data protection requirements. The families in this class are organized into four groups, i.e., *User data protection security function policies*, *Forms of user data protection*, *Off-line storage import and export*, and *Inter-TSF communication*. A developer selects the required families for a target system. We define *Access Control functions* (FDP_ACF) as an SFP table according to the *Access control Policy* (FDP_ACC) of LUMINOUS BBS in this case.

We explain the mapping rule that combines the SFP with both data and actions in the UML RA model written in activity diagrams and a class diagram.

A rule in SFP is defined by the relation between a subject, an operation, and a target object. A subject carries out an operation. The RA model consists of actors, use cases (activity diagrams), classes, and actions in the activity diagram. Figure 4 shows that in the mapping rule a subject corresponds to an actor, an object corresponds to a class, and an operation corresponds to an action in a use case. Some new security attributes need to be defined against both the assets extracted in Section III and the subjects who carry out the controlled operations, because SFP controls action flows by a rule based on security attributes. Table I shows that an SFP can be defined by actions and data in the target system, according to the correspondence.



Correspondence	
Subject	Actor
Operation	Action
Object	ObjectNode:Class

Fig. 4. Mapping Rule between SFP and RA Model

C. *Defining SFP According to the Selected Components and the Dependency on the CC*

A developer selects a subset of the operations for a subset of the objects to be controlled based on the results of misuse case analysis. That means that he/she has selected a subset access control component FDP_ACC.1 on an access control policy. Next, he/she defines a security attribute based access control function, in accordance with the family FDP_ACF. The developer defines a security attribute based access control policy, which is a kind of SFP for the target system. Subjects, objects, and operations in an SFP correspond to actors, classes, and actions, respectively. Therefore, these elements are extracted from the activity diagrams and a template table based on the dependencies of the components is automatically generated.

Table I shows that we have to consider each rule for each component, because FDP_ACF.1 has dependencies such as those shown in Figure 5.

Table I shows the access control policy for the BBS. First, the developer sets security attributes for the specified assets. The developer analyzes an SFP for security attributes, according to the dependency on security components as shown in Figure 5, based on access control as follows. The resulting SFP in Tables I and II show the details of the rules.

The developer defines security attributes against the selected assets based on FDP_ACF.1 and defines the rules to explicitly authorize or deny access to them. For example, if rule A is:

At the start of executing an action *download Attachment, attachment.public/private* == public || *contributer.role* == *student_id.*

The developer defines that the default values of security attributes are either permissive or restrictive in nature as appropriate based on FMT_MSA.3. For example, ruleB2 is

At the end of executing an action, a*ttachment.public/private* == private.

The developer decides that the target system has a role that can manage the specified security attributes based on FMT_MSA.1. Then, he/she decides which authorized users have that role based on FMT_SMR.1. A teacher has this role in this case. Moreover, he/she decides on some management functions that are performed by the specified role based on FMT_SMF.1. For example, an action *update public/private to private* is introduced. When new functions are introduced the developer must investigate whether new security policies related to them need to be introduced as well.

TABLE I. PART OF SFP OF LUMINOUS BBS

Subject		Object		Operation		Rule		
Actor	Securty Attribute	Class	Security Attribute	Use Case	Action	FDP_ACF.1	FMT_MSA.3	FMT_MSA.1
Student	role =Student ID	. . .						
		Topic	public/private	post a question	create topic		rule B1	
		Date		post a question	get currentDate			
		Contibutor	role	post a question	get contributor			
		Content		read a topi_student	get content			
		Attachment	public/private	read a topi_student	download attachment	rule A		
				post a question	create attachment		rule B2	
Teacher	role =primary Charge	. . .						
		Topic	public/private	answer a question	select specified topic			
				answer a question	update the topic			
				update topic openness	change attribute of topic(public)			rule C1
				update topic openness	change attribute of topic(private)			rule D1
		Date		answer a question	get currentDate			
		Contributor	role	answer a question	get contributor			
		Content		read atopic_teacher	get content			
		Attachment	public/private	read atopic_teacher	download attachment			
				answer a question	create attachment		rule B3	
				update attachment openness	change attribute of attachment(public)			rule C2
				update attachment openness	change attribute of attachment(privatec)			rule D2

The family FIA_UID defines the conditions under which users shall be required to identify themselves before performing any other actions that are to be mediated by the target system that require user identification. The developer has nothing to do in this case.

A blank in the table denotes that the combination is unrelated to security requirements.

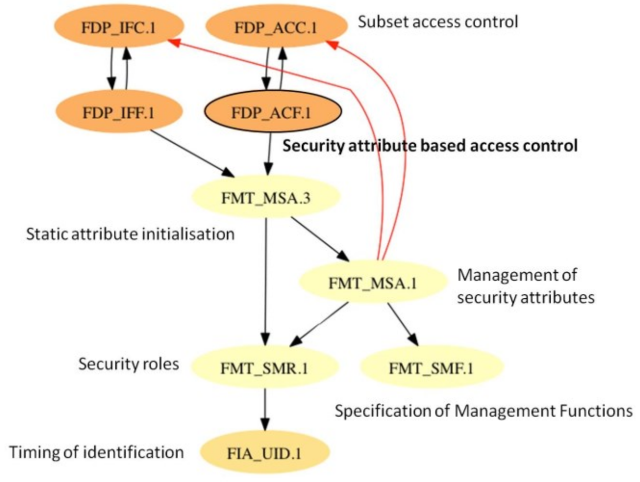

FDP_IFC.1
FDP_ACC.1
Subset access control
FDP_IFF.1
FDP_ACF.1
Security attribute based access control
FMT_MSA.3
Static attribute initialisation
FMT_MSA.1
Management of security attributes
Security roles
FMT_SMR.1
FMT_SMF.1
Specification of Management Functions
Timing of identification
FIA_UID.1

Fig. 5. Dependency on FDP_ACF.1

TABLE II Access Control Rules

rule A	At the start of executing the action, attachment.security_attrubute == public||contributor.role ==student_id
rule B1	At the end of executing the action, topic.security_attribute == private
rule B2	At the end of executing the action, attachment.security_attribute == private
rule B3	At the end of executing the action, (topic.security_attribute == public implies attachment.security_attribute == public|| attachment.security_attribute == private) && (topic/security_attribute == private implies attachment/security_attribute == private)
rule C1	At the start of executing the action, topic.security_attribute == private implies at the end of executing the action, topic.security_attribute == public
rule C2	At the start of executing the action, attachment.security_attribute == private implies at the end of executing the action, attachment.security_attribute == public
rule D1	At the start of executing the action, topic.security_attribute == public implies at the end of executing the action, topic.security_attribute == private
rule D2	At the start of executing the action, attachment.security_attribute == public implies at the end of executing the action, attachment.security_attribute == private

D. *Defining State Machine Model of Security Attributes*

The state transition of each security attribute is defined as a state machine model, by discriminating states for each security attribute. Figure 6 shows a state machine for a topic that is one of the assets to be protected from malicious students. There are three states to be distinguished as security attributes and three events corresponding to actions in the activity diagram that are trigger events which cause each transition.

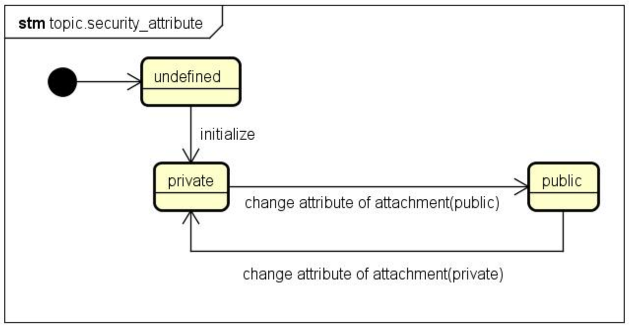

stm topic.security_attribute
undefined
initialize
private
public
change attribute of attachment(public)
change attribute of attachment(private)

Fig. 6. State Machine Diagram of an Asset.

V. Security Requirements Verification

This section explains how to transform the RA model and specified security requirements SFP from UML to UPPAAL [10], and how to generate the query expressions.

A. *Model Checking Tool UPPAAL*

Figure 7 shows that the UPPAAL model consists of several locations and transition arrows among them. A location expresses a system state and a transition arrow indicates several conditions named *Guard* and a sequential processing event that occurs during it named *Update*. Figure 7 shows START, LOC1, and LOC2 as the names of each location. “i1==0” and “i1>0” are the *Guard* expressions and “flg=true” and “flg=false” represent the *Update* expressions.

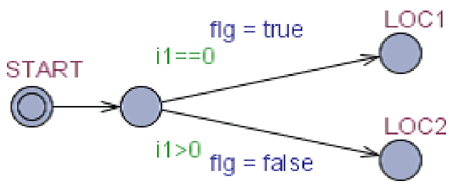


Fig. 7. Basic Components of the UPPAAL Model

B. *Verification Process Overview*

The RA model includes all the use cases for a target system and a navigation model to integrate them. Figure 8 shows the entire process to transform UML models into UPPAAL models. Query expressions are generated from the rules shown in Table II.

First, each activity diagram corresponding to a use case is transformed into one system model in UPPAAL. All nodes, such as action, object, decision, merge, start, end, etc., in an activity diagram, are transformed into locations in UPPAAL. The control flow and data flow are each transformed into transitions, except for the update actions of each security attribute.

Assuming that an update action is *U*, it is transformed into a transaction sequence with three locations. The first location represents a pre-state before calling action *U*, and the second location represents an update state. The third location represents a post-state after the update. The first transiti on flow has a synchronization channel named “c_U!” and the second transition flow has a synchronization channel named “r_U?”, where “c” denotes “call” and “r” denotes “return,” respectively.

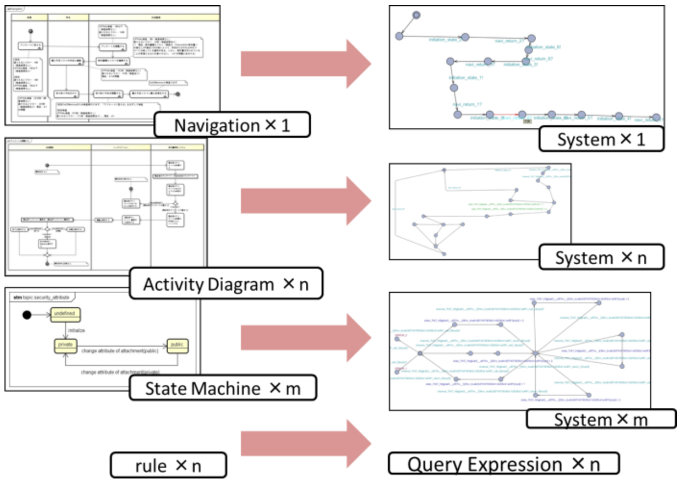


Fig. 8. Transformation process from UML to UPPAAL

These synchronization channels coordinate with other channels in a system that is being transformed from a state machine diagram of the corresponding object class. The corresponding object is an objective word of the *updating* action in this case.

Every action in this model that defines an *event* in a state machine diagram is transformed into a transition of three locations with channel synchronization.

A navigation model integrates all activity diagrams according to the pre-conditions and post-conditions, which are a combination of several labels added to the start or end nodes in each activity diagram. All system models transformed from the activity diagrams are integrated as a UPPAAL model according to these conditions.

C. *Security Requirements Verification*

All system models that are transformed from UML are integrated by using a mechanism of channel synchronization and several variable declarations specified by the SFP. Figure 9 shows that each variable corresponds to an attribute of an object, including security attributes that appear in the state machine diagram.

Rules in the SFP, shown in Table II, are expressed only by the specified use case or action and the attributes. These components are transformed into locations or variables in the UPPAAL model, therefore, a query expression can be automatically generated.

The RA model is verified by the following procedure. First, the validity of the whole model is checked.

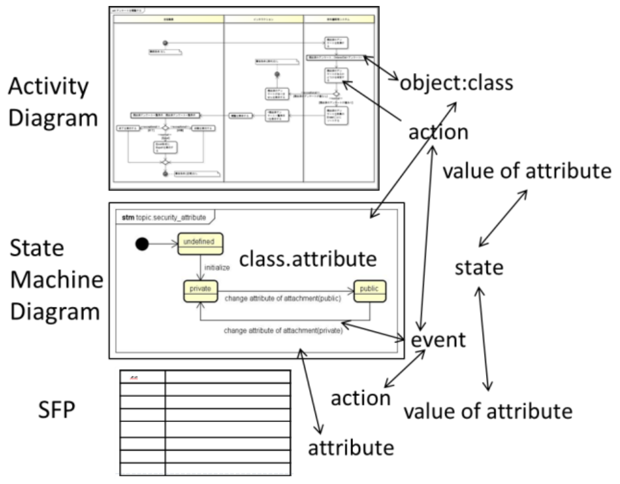


Fig. 9. Correspondence between Components

A) All control flows which are defined in the activity diagrams are validated by checking the reachability of all locations in the UPPAAL model. Reachability means that the specified state will be reached at some point in time. Each query expression is expressed by the following expression and the reachability to the specified location is checked. Each location name is specified by an activity name, ACT_nameXX, and a location name, location_nameXXXX.

```
E<> ACT_nameXX.location_nameXXXX
```

If this query expression is not satisfied, the developer needs to revise the activity diagram accordingly.

B) There is a possibility that a location exists which is never passed through in the entire system model.

Each query expression is represented by the following expression. This means that all processes never generate a deadlock state, except for the end location of the activity diagram.
ACT_nameXX.END means the end location of an activity diagram named ACT_nameXX.

```
A[] not (deadlock && !ACT_nameXX.END)
```

If this query expression is not satisfied, the developer decides whether or not the process is improbable in the application. Then he/she revises the activity diagram as necessary.

Next, we check the basic behavioral property of a security attribute defined by a state machine diagram corresponding to a class.

C) All states of a class defined by the state machine diagram are validated in the activity diagrams, by checking the reachability of the relevant location in the UPPAAL model. Each query expression is represented by the following expression and the reachability to the specified location corresponding to the state is checked. For example, “state_topic(0)” represents the state named “undefined” in the state machine diagram of the “Topic” class. “Topic_init” represents a location corresponding to the action “initialize”. The other states are checked in the same manner. Each state and location name is defined automatically when it is transformed into the UPPAAL model.

```
E<> state_topic(0).Topic_init
E<> state_topic(1).Private_to_Public
E<> state_topic(2).Public_to_Private
```

If this query expression is not satisfied, the developer needs to add an initializing action to the relevant activity diagram.

Next, we check the security requirements defined in Table II.

For example, rule C1 is described as follows.

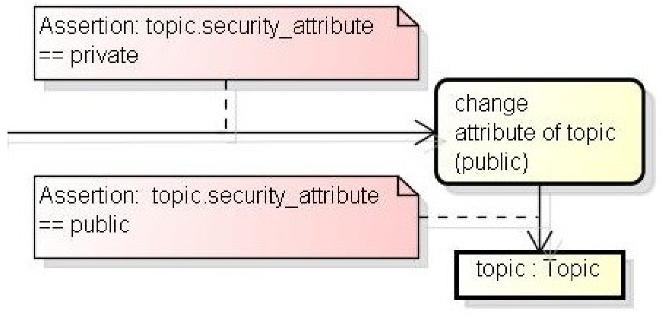
At the start of executing the action, topic.security_attribute == private implies at the end of executing the action, topic.security_attribute == public

In this case, the sentences “At the start of executing the action” and “at the end of executing the action” are translated into locations by adding notes with the format shown in Figure 10.

The word “topic.security_attribute” is translated into a variable “state_topic[0]” and the attribute values “public” and “private” are translated into integers 3 and 2, respectively.

The result is that the query expression corresponding to rule C1 is as follows.

```
A[] (ACT03.Assert01  imply state_topic[0]==3)
A[] (ACT03.Assert02  imply state_topic[0]==2)
```



Assertion: topic.security_attribute == private
change attribute of topic (public)
Assertion: topic.security_attribute == public
topic : Topic

Fig. 10. Notes in Activity Diagram

VI. Related Work

Several researchers have proposed formal approaches to verify specified features in the early stages of software development. Yatake [11] verified that all object states satisfy the invariant conditions between collaborative object behaviors, by using a theorem-proving system. However, it requires a large quantity of strict definitions to clarify all the actions and data relating to the invariant. It is generally difficult to perform such strict work during a dynamic phase such as requirements analysis.

Stepwise specification refinement is important to conduct in the early stages of software development, by checking several verifiable features. Choi [12] proposed a verification method of the consistency between the page transition specification on a web-based system and the flow chart defining the process streams. We also propose non-functional features for enterprise systems, such as security requirements for specified functional requirements. Moreover, we can automatically generate the query expressions.

Achenbach [13] compared the abstraction techniques in various model checking tools and applied these tools to real-world problems. For example, the open/close behavior of the file I/O stream was modeled using the transition between states such as open, close, and error. This approach is very similar to ours. However, unlike our approach, this paper does not discuss how to verify such properties that are strongly related to the target application as access control conditions.

Several researchers have proposed support methods to effectively use model checking tools [14, 15, and 16].

Trcka [14] proposed a method to verify the nine predefined query expressions using a Petri net, which can specify behaviors such as read, write, and delete. This study may be similar to our method; however, because query expressions depend on the properties specified by state machine diagrams, our method can be extended to verify the other properties as well.

Several studies [15, 16] have proposed methods to transform UML models into the process or protocol meta-language (PROMELA), for use with the model checking tool SPIN. However, because developers need to directly operate the model checking tool, they are required to have knowledge of both UML and SPIN. Our approach has the advantage that UML2UPPAAL can be used by developers with only UML knowledge.

VII. Conclusion

This paper proposed a verification method for requirements specifications in UML at the beginning of development with a model checking technique. The results are more reliable than ad hoc analysis, because security requirements can be verified according to the CC.

We also developed a support tool to verify security requirements by transforming a requirements specification written in UML into a finite automaton in UPPAAL. The tool has an advantage in that UML developers can benefit from the UPPAAL model checking technique without the need for additional knowledge. We plan to apply our method to verify the other security requirements for the entire LUMINOUS system.

References

[1] Common Criteria, ” CC/CEM v3.1 Release4 ”, http://www.commoncriteriaportal.org/cc/

[2] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard. G., *Object-oriented software engineering: A usecase driven approach*, Addison-Wesley Publishing, 1992.

[3] S. Ogata, and S. Matsuura, "A UML-based Requirements Analysis with Automatic Prototype System Generation," Communication of SIWN, Vol.3, pp.166-172, 2008.

[4] S. Ogata. and S. Matsuura, "A Method of Automatic Integration Test Case Generation from UML-based Scenario," WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 4, Vol.7, pp.598-607,2010.

[5] Y. Aoki, S. Ogata, H. Okuda and S. Matsuura, Quality Improvement of Requirements Specification Using Model Checking Technique, Proc of ICEIS 2012, Vol.2,pp401-406, 2012.

[6] OMG," UNIFIED MODELING LANGUAGE", http://www.uml.org/

[7] LUMINOUS, https://lmns.sayo.se.shibaura-it.ac.jp/

[8] Sindre, G and Opdahl, A. L. "Eliciting security requirements with misuse cases", Requirements Engineering Journal, Vol.10, No.2 (2005).

[9] astah, http://astah.net/

[10] UPPAAL, http://www.uppaal.com/, 2010.

[11] K. Yatake, T. Aoki and T. Katayama, "Collaboration-based verification of Object-Oriented Models", Computer Software, Vol.22, No.1, 2005, pp.58-76. (in Japanese)

[12] E. Choi, T. Kawamoto, and H. Watanabe, "Model Checking of Page Flow Specification", Computer Software, Vol.22, No.3, 2005, pp.146-153. (in Japanese)

[13] M. Achenbach and K. Ostermann, "Engineering Abstractions in Model Checking and Testing", Source Code Analysis and Manipulation, Proc. of .SCAM '09.,2009, pp.137-146

[14] N. Trcka, Wil M. Aalst, and N. Sidorova ., "Data-Flow Anti-Patterns: Discovering Dataflow Errors in Workflows," Proc. of the CAiSE 2009, 2009, pp.425-439.

[15] P. Bose, "Automated translation of UML models of architectures for verification and simulation using SPIN," Proc. of the ASE, 1999, pp.102-109.

[16] L. Jing, L. Jinhua, and Z. Fangning, "Model Checking UML Activity Diagrams with SPIN," Proc. of the CiSE 2009, 2009, pp.1-4.