Model-Driven Security Patterns Application Based on Dependences among Patterns

Yuki Shiroma, Hironori Washizaki,
Yoshiaki Fukazawa
Dept. Computer Science and Engineering
WASEDA University
Tokyo, Japan
shiroma1986@moegi.waseda.jp, washizaki@waseda.jp,
fukazawa@waseda.jp

Atsuto Kubo, Nobukazu Yoshioka
Dept. Information Systems Architecture Science
Research Division
National Institute of Informatics
Tokyo, Japan
kubo@nii.ac.jp, nobukazu@nii.ac.jp

Abstract— **The spread of open-software services through the Internet increases the importance of security. A security pattern is one of the techniques in which developers utilize security experts' knowledge. Security patterns contain typical solutions about security problems. However there is a possibility that developers may apply security patterns in inappropriate ways due to a lack of consideration on dependencies among patterns. Application techniques of security patterns that consider such dependencies have not been proposed yet. In this paper, we propose an automated application technique of security patterns in model driven software development by defining applications procedures of security patterns to models as model transformation rules with consideration for pattern dependencies. Our technique prevents inappropriate applications such as the application of security patterns to wrong model elements and that in wrong orders. Therefore our technique supports developers apply security patterns to their own models automatically in appropriate ways.**

Keywords-component; Security Patterns; Model Driven Development;UML;ATL;

I. INTRODUCTION

The spread of open-software services through the Internet highlights the growing importance of software security.

It is imperative to consistently improve security because security is at risk in security problems at every phase of development. How to do so is non-straightforward. Moreover, ever when security has been achieved, it is necessary to consider the trade-off with other quality characteristics, and a lot of development experience is required to make an appropriate judgment. Security patterns have been proposed to assist developers in handling security concerns. By using security patterns, software developers can utilize security specialists' knowledge by using security patterns.

Moreover, security patterns have dependences among patterns. So developers should decide the sequences of the pattern application considering dependences. If this dependency is not considered, there's a chance that security patterns could be incorrectly applied and the security of the entire system would suffer. However, as far as we know, no application support technique that considers dependences between patterns has been proposed.

To remedy this situation, we propose a security pattern application technique considers dependences between patterns. Patterns are applied by making model transformations. When a security pattern is applied, our proposed system leaves a mark about its application in the model, and subsequent security patterns are applied at this mark left previously. This enables consecutive applications of security patterns.

II. BACKGROUND FOR THE PROPOSED TECHNIQUE

This section describes the ideas behind model-driven development and security patterns. In addition, it discusses the problem in regard to dependences between security patterns.

A. Model Driven Development(MDD)

Model driven development is a methodology that builds software around a model [7]. The developers translate an abstract model into a more concrete model (i.e. UML). The developer can obtain the source code semi-automatically by repeating the transformation into a more concrete model. Moreover, there are various model transformations, e.g., model merging and model marking.

B. Security Patterns

Security patterns describe problems which frequently occur and the core of the solution to each problem. The advantage of the security patterns is that they utilize the knowledge of security specialists. Security patterns provide guidelines for improving confidentiality, integrity, and availability in the software development. Security patterns are described in terms of a Structure, Context, Problem, Solution, and Consequences [2].

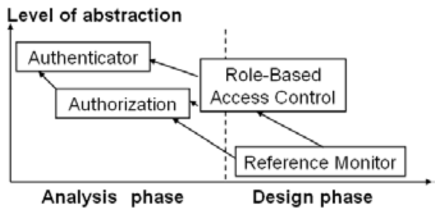


Figure 1. Example of dependences among security patterns

Security patterns have dependences among patterns and when developers are to apply security patterns consecutively, they should be applied consistently throughout the development process. Thus it is very important to consider the dependences between security patterns in the entire development process. Figure 1 shows an example of dependences between security patterns. For instance, "Authenticator <- Authorization" means that the Authenticator pattern should be applied before applying the Authorization pattern. The sequences of the possible application are as follow.

- Authenticator, Authorization, RBAC, Reference Monitor
- Authenticator, Authorization, Reference Monitor
- Authenticator, RBAC, Reference Monitor

Notice that the three problems.

P1. Possibility of the wrong application of the security pattern if the dependences among patterns is not considered

P2. Possibility of the security pattern application in an incorrect part of the model

P3. Huge cost of time and labor

However, the known supports for developers include only classification [3] and unit security pattern application support [5].

III. Proposed Technique

This section describes the method for making transformation rules, the use of the proposed system, and the solution to the problems. There are three solutions to the above problems correspondingly named S1, S2, and S3.

S1. Automatization of model transformation

S2. Making of a security pattern transformation rule library

S3. Marks of the application result are left in the class, and use them to apply following security patterns.

S3 corresponds to the solution of P1 in the preceding section, S1 and S3 correspond to the solution of P2, and S1 and S2 correspond to the solution of P1.

A. *Method of Describing Transformation Rules*

A security specialist describes transformation rules derivable from a security pattern by using ATL (ATLAS Transformation Language)[8]. The transformation rules each consist of a pre-condition, an argument and operation.

A pre-condition is the assumption of the security pattern application. The definition of a pre-condition varies according to the presence of an existing pattern upon which the to-be-applied security pattern depends.

If no pattern on which the applied security patterns depends exists, security specialists define which part corresponds to the role of the security pattern in the model. If such a pattern exists, security specialists place a mark in the model that indicates that the existing pattern is a pre-condition for applying the new pattern. Our technique supports consecutive applications of security patterns considering the dependence among patterns by defining the output of the previous model transformation as a pre-condition in the subsequent model transformation.

Security specialists should determine the dependences among patterns by referring to the "Related patterns" section in the security pattern catalog. Moreover, there is a possibility that the dependence could be obtained by looking at the problem and the context of the applied pattern, and also by looking at the context, solution, and consequence sections of all the other patterns.

For instance, it is described that "The authenticated user, represented by processes running on its behalf, and is then allowed to access resources according to their rights" (p. 323) in the context of the Authenticator pattern in [1]. On the other hand it is described that "Any environment in which we have resources whose access needs to be controlled" (p. 245) in the context of the Authorization pattern. It can be judged that a relation exists between these two patterns because their context sections resemble each other with regard to controlling access to the protected property, even though this relationship is not explicitly written in either pattern. Moreover, the description "What the attested user accesses the protection property by the authority is permitted" shows that we should authorize after authenticating. Therefore, we can see that the Authenticator pattern should be applied before applying the Authorization pattern.

The argument is a parameter that the software developer should input (i.e., the name of the class that corresponds to the role of the applied security pattern).

The operation maps a set of classes and relations among classes in the model when the pattern is applied. Security specialists should look for common roles between the applied pattern and to-be-applied pattern in their structural description.

For instance, the Subject role of the Authenticator pattern is described in [1] as, "A Subject, typically a user, and requests access to system resources" (p. 324). Moreover, the Subject role of the Authorization pattern is described in [1] as, "The Subject class describes an active entity that attempts to access a resource (Protection Object) in some way" (p. 246). The Subject roles of the two security patterns are identical because the descriptions of the Subject role are very similar. In a word, when there is the dependence between the two patterns, a common role is the basis for deriving the transformation rule of each pattern. Therefore, security specialists should describe a transformation rule whereby the Subject role of the Authenticator pattern corresponds to the role of the Authorization pattern. A security pattern is applied by transforming the model by using the above-mentioned transformation rule.

B. *Example Description of Transformation Rules*

We offer an example of describing the transformation rules when applying an Authorization pattern.

A pre-condition is that the class with the stereotype described 'Authenticator.Subject' exists. This stereotype indicates where the Authenticator pattern, on which the Authorization pattern depends, is applied in the model.

The argument is a class name that corresponds to the Protection Object role of the Authorization pattern.

The operation has the following five steps.

- Add the stereotype ‘Authorization.ProtectionObject’ to the Class that corresponds to the Protection Object role that the developer inputs as an argument.
- Add the class that corresponds to the Right role.
- Add the relation between the class that corresponds to the Subject role and the class that corresponds to the Right role.
- Add the relation between the class that corresponds to the Protection Object role and the class that corresponds to the Right role.
- Remove the relation between the class that corresponds to the Protection Object role and the class that corresponds to the Subject role.

To do the above-mentioned mapping, security specialists describe the transformation rules by using ATL. Figure 2 shows part of a transformation rule of the Authorization pattern described in ATL. The isProtOb function of the first line in Figure 2 judges whether the character string of the class name that corresponds to the Protection Object role that the software developer inputs correspond to the class name in the model. The hasStereotype function of the fourth line judges whether the class in the model has the stereotype described 'AuthenticatorSubject'.

```
helper context UML!Class def:isProtOb() : Boolean =
if self.name = thisModule.ProtObName
 then true else false endif;
Helper context UML!Class def:hasStereotype(stereotype : String) :
Boolean = self.stereotype->collect(s | s.name)->includes(stereotype)⋯
rule ProtectionObjectClass {
 from s : UML!Class (s.isProtectionObject()⋯
  stereotype <- stereotypePO),
  stereotypePO : UML!Stereotype (
   name <- 'Authorization.ProtectionObject',   ⋯
rule SubjectClass {
 from s : UML!Class (s.hasStereotype('Authenticator.Subject')) ⋯
```

Figure 2. Part of the transformation rule of the Authorization pattern

Marking by the stereotype was chosen as the form of the model transformation in to show where a security pattern is applied.

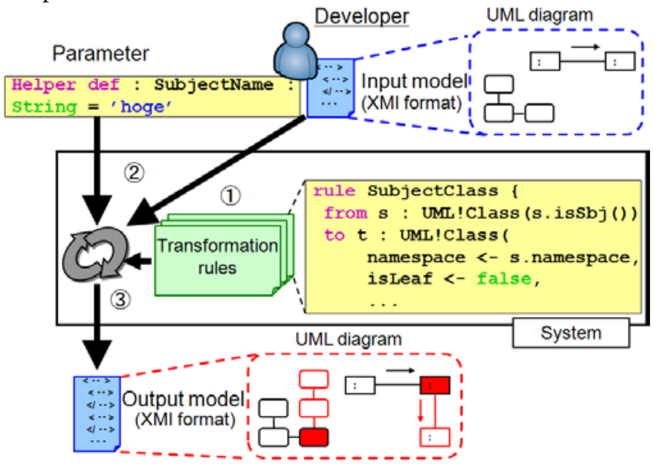
The reason for choosing ATL as the model transformation language is that it is easy for software developers to understand and it can easily be extended because it is based on Queries/Views/Transformations (QVT), which is a standard model transformation.

C. *Application Procedure*

The system transforms a UML model (XMI format) inputted by software developers and the security pattern is applied by making a model transformation. Figure 3 shows the image of the proposed system. The security pattern is applied as follows.

1. The developer selects the security pattern.
2. The developer inputs the model and the parameters to the proposed system.
3. The system transforms the model and outputs the model with the applied security pattern applied.

The system deals with two models as input and output: Class diagram and Communication diagram. These are described in XMI format. The transformation rules, once described can be reused. Consecutive application of security patterns considering the dependences among patterns becomes possible by using the obtained output model as the input model for the subsequent transformation. By using marks, it can be automatically judged whether the class and the pattern role are the same.



```
Helper def : SubjectName :
String = 'hoge'
```

```
rule SubjectClass {
 from s : UML!Class(s.isSbj())
 to t : UML!Class(
     namespace <- s.namespace,
     isLeaf <- false,
     ...
```

Figure 3. Entire image of the proposed system

D. *The Distribution of Security Patterns*

The proposed technique can deal with security patterns that are described the structure. The proposed technique deals with 27 security patterns in [1]. So far, 19 security patterns in [1] cannot be dealt with because their structures are not described.

IV. EXAMPLE

We shall consider a Patient’s Information Management System (PIMS) in a hospital as an example of applying the proposed technique. Figure 4 shows the use-case diagram of the PIMS. The following two security requirements are necessary for the PIMS.

SR1. Only hospital employees can access the PIMS. Confidentiality is thus maintained.

SR2. The user of the PIMS can only do the use case with the allocated authority. Confidentiality is thus maintained.

The PIMS faces two problems in regard to meeting the security requirements.

The first problem is that there is no structure to judge if the user is an employee or not. A third party could thus pose as an employee in order to steal patient information and sell it (misuse case 1).

The second problem is that everyone related to the hospital has read and write access to the patient's information. Even if the first problem is solved, the second problem remains. A potential problem is that someone could illegally rewrite a patient’s examination results (misuse case 2).

The class diagram of the PIMS is shown in Figure 5, and part of XMI of the class diagram is shown in Figure 6. Two security patterns are applied as a solution of the above-mentioned threats. First, the Authenticator pattern concerning the authentication is applied. Then, the Authorization pattern concerning authorization is applied.

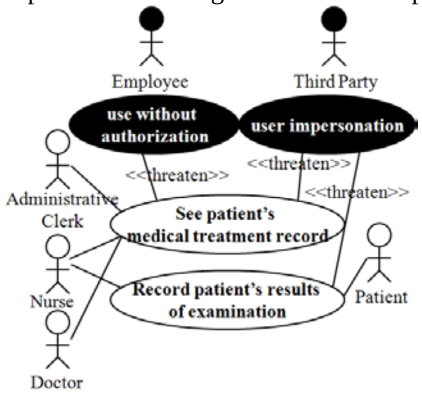


Figure 4. Use-case diagram of the PIMS.

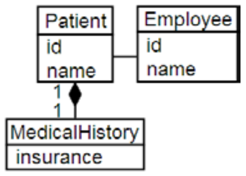


Figure 5. Class diagram of the PIMS

```
<UML:Class xmi.id = 'a2' name ='Employee' ···
```

Figure 6. Part of XMI in the class diagram

A. Application of the Authenticator Pattern

After selecting the ATL file in which the Authenticator pattern is described, the developer inputs the model and the parameters to the system. If the developer inputs “sbjName =‘Employee'”, the system decides that the Employee class corresponds to the Subject role of the Authenticator pattern and applies the Authenticator pattern. A stereotype is added to indicate that the Employee class corresponds to the Subject role. Figure 7 shows the class diagram of the Authenticator pattern and Figure 8 shows the class diagram after the Authenticator pattern has been applied. Figure 9 shows the XMI for the class diagram after the Authenticator pattern has been applied.

The system judges that the Employee class corresponds to the Subject role of the Authenticator pattern and applies the Authenticator pattern to the model because developers inputted the parameter “sbjName =‘Employee'”. At this time, a mark is applied to indicate that Employee class corresponds to the Subject role by adding the stereotype to the Employee class.

Because of the authentication structure is added to the model by applying Authenticator pattern, countermeasures against a malicious third party disguised as a user were given. However, the problem that a malicious employee can access patient information remains because every employee is granted access to the information. To combat this problem, an Authorization pattern is required.

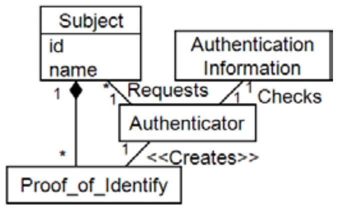


Figure 7. Class diagram of the Authenticator pattern

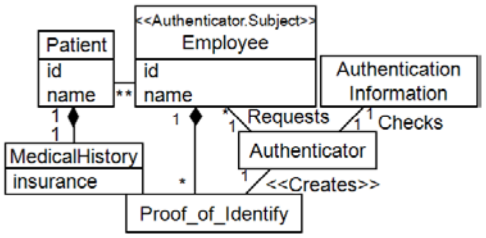


Figure 8. Class diagram after the Authenticator pattern is applied

```
<UML:Class xmi.id = 'a2' name ='Employee'  ···
   <UML:Stereotype xmi.idref = 'a3'/>     ···
 <UML:Stereotype xmi.id = 'a3' name =
'Authenticator.Subject'···
<UML:Class xmi.id = 'a10' name =
'Authenticator'···
```

Figure 9. XMI of the class diagram after the Authenticator pattern

B. Application of the Authorization Pattern

After selecting the ATL file in which the Authorization pattern is described, the developer inputs the model and the parameters to the system. If the developer inputs “protObName =‘Patient”, the system judges that the Patient class corresponds to the Protection Object role of the Authorization pattern and applies the Authorization pattern.

Moreover, because the Employee class that corresponds to the Subject role applies the stereotype ‘Authenticator.Subject' when the Authenticator pattern was applied, the system judges that the Employee class corresponds to the Subject role of the Authenticator pattern and to the Subject role of the Authorization pattern.

The Authorization pattern is applied to the model through the above process. Also, a stereotype is added to indicate that the Patient class corresponds to the Protection Object role and to indicate that the Subject role corresponds to the Employee class of the Authorization pattern. Figure 10 shows the class diagram of the Authorization pattern, and Figure 11 shows the class diagram after the Authorization pattern has been applied.

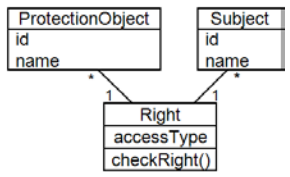


Figure 10. Class diagram of the Authorization pattern

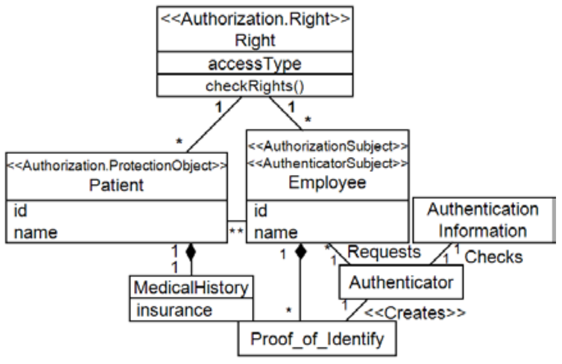


Figure 11. Class diagram after the Authorization pattern is applied

The PIMS had two security problems in that there was no authentication and authorization structure. To solve them patterns were applied. The countermeasure against user impersonation was taken by applying Authenticator pattern correctly and the countermeasure against use without authorization was taken by applying Authorization pattern correctly.

V. Evaluation

Here, we weigh the merits of the proposed security pattern application against those of the manual security pattern application. The comparison shall be in terms of the number of work steps and the time required for the security pattern application.

There are five work steps calculated is the following five: (1) addition, deletion of the class, (2) addition, deletion of the relation, (3) input the name of the class, (4) automatic model transformation, and (5) input an argument in the model transformation.

The time required was assumed to be the mean value of the times required to apply security patterns measured in an experiment involving six senior year university students who had experience with a UML modeling tool.

Figure 12 shows the times required for the security pattern application and Table I lists the number of work steps. The proposed technique saves 71% of the time spent manually, and it reduces the number of steps by more than 50%.

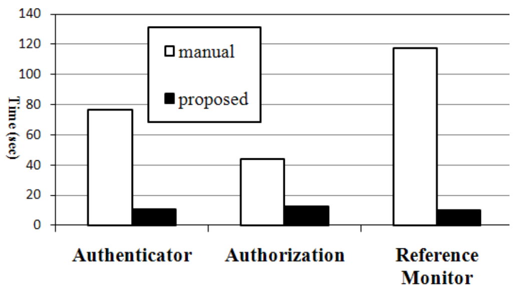


Figure 12. Time required for the security pattern application

TABLE I. Number of work steps

Method	Security Pattern		
	Authenticator	*Authorization*	*Reference Monitor*
Manual	7	4	5
proposed	2	2	1

VI. Related Work

Yu et al. [3] proposed a security pattern technique that transforms the i* model using ATL. Moreover, Horvath proposed a technique for converting a model using the petri net. [5] Ours is different from these existing techniques because its model transformations use UML and its security patterns are written in ATL.

VII. Conclusion

The proposed technique enabled automatic consecutive applications of security patterns that depend on each other by concretely establishing a method of describing security pattern transformation rules and by marking the point in the model at which it was transformed.

Although other patterns besides the ones discussed here can be applied, their dependences may not be as obvious as illustrated here.

Our future work will include the following four tasks.

- Cover all 27 security patterns that can be treated by the proposed technique.
- Ensure the security of the entire system by using security patterns that strictly describe the security properties.
- Derive dependences between patterns from the pattern documents by applying Kubo's technique [4]
- Quantitatively evaluate the accuracy of the security pattern applications.

References

[1] Schumacher, M et al.:Security Patterns, Wiley, 2006

[2] Yoshioka, N et al.: A Survey on Security Patterns, Progress in Informatics, no.5, pp. 35-47. 2008

[3] Yu, Y et al.:Enforcing a Security Pattern in Stakeholder Goal Models, ACM Proc. ACM workshop on Quality of protection (QoP'08), 2008

[4] Kubo, A et al.: Extracting Relations among Embedded Software Design Patterns, Journal of Integrated Design and Process Science (SDPS), pp.39-52, vol.9, no.3, 2005

[5] Horvath, V. Dorges, T.:From Security Patterns to Implementation Using Petri Nets, ACM, 2008

[6] Schumacher, M. Roeding, U.:Security Engineering with Patterns, LNCS2754, pp. 121 – 140,2003

[7] Frankel, D.:Model Driven Architecture, Wiley,2003.

[8] Eclipse.org. ATL Project, http://www.eclipse.org/m2m/atl/

[9] Jurjens, J.: Secure Systems Development with UML, Springer, 2004

[10] Lodderstedt, T et al.:SecureUML: A UML-Based Modeling Language for Model-Driven Security, In Proceedings of the 5th International Conference on the Unified Modeling Language, pp. 426-441,2002

[11] Sindre, G et al.: Elicting security requirements with misuse cases, Requir.Eng., vol 10, no.1, pp.34-44, 2005