Enforcing a Security Pattern in Stakeholder Goal Models

Yijun Yu
The Open University
UK
Y.Yu@open.ac.uk

Haruhiko Kaiya
Shinshu University
Japan
kaiya@cs.shinshu-u.ac.jp

Hironori Washizaki
Waseda University
Japan
washizaki@waseda.jp

Yingfei Xiong
University of Tokyo
Japan
xiong.yingfei@gmail.com

Zhenjiang Hu
GRACE Center, NII
Japan
hu@nii.ac.jp

Nobukazu Yoshioka
GRACE Center, NII
Japan
nobukazu@nii.ac.jp

ABSTRACT

Patterns are useful knowledge about recurring problems and solutions. Detecting a security problem using patterns in requirements models may lead to its early solution. In order to facilitate early detection and resolution of security problems, in this paper, we formally describe a role-based access control (RBAC) as a pattern that may occur in stakeholder requirements models. We also implemented in our goal-oriented modeling tool the formally described pattern using model-driven queries and transformations. Applied to a number of requirements models published in literature, the tool automates the detection and resolution of the security pattern in several goal-oriented stakeholder requirements.

Categories and Subject Descriptors: D.4.6 Security and Protection: Access Control

General Terms: Algorithms, Design, Languages, Security.

Keywords: Security Patterns, RBAC, Goal Models, Model Transformations.

1. INTRODUCTION

Patterns have been proposed to categorise knowledge of recurring problems in software design and give advices on how to resolve them [1]. It has been widely accepted that a pattern language should also be created to categorise security problems in software designs [2].

Fixing errors earlier in requirements costs much less than fixing accumulated errors in design and implementation [3]. For security problems, therefore, one may ask a relevant question "Can we detect security problems and even resolve them early before it is too late?"

This question has led to active research on the representation and analysis of security requirements [4, 5, 6]. Work in this area assumes that security requirements can be elicited by arguing thoroughly about vulnerability in existing requirements models, such as trust assumptions [7], anti-goals [8], misuse cases [9], abuse frames [10], risk analysis [11, 12], etc. Yet little has been done to suggest systematical changes in requirements models to resolve these vulnerabilities. Partly due to the fact that it is impossible to detect and resolve once-for-all the vulnerabilities, especially when not all problematic trust assumptions or anti-goals may be detected. Therefore, a sound protection mechanism should be open to incorporate new security patterns.

Capturing lessons learnt from the past as well-known security patterns, one may still need to enforce them to a requirements model for a new software project. In this paper, we propose to represent security patterns formally on basis of existing requirements modeling languages, such that an analysis tool can be developed to detect and resolve security problems in the modelled requirements. For such automated analysis, it is a reasonable assumption for stakeholder requirements to be modelled using a formal language: once requirements have been modelled as such, the tool could guarantee all instances of the security pattern can be detected and necessary changes can be suggested.

To illustrate, we tried a single security pattern for recurring problems in role-based access control. Although it should be possible to define the pattern on different formal requirements modeling languages, we chose the goal-oriente requirements modeling language i*/Tropos [13, 14], for two reasons. One, it is widely used in early requirements engineering which leads to many published models in the literature. Second, we have developed the support using Eclipse modeling framework, which enables the techniques presented here.

Our main contribution is to show that such formally defined security pattern can be directly used to detect and resolve the security problem on requirements models published in the literature. The key techniques used are model-driven query and transformation [15] and model-driven development of requirements models [16], both are integrated in our requirements engineering tools[1].

The remainder of the paper is organised as follows. Section 2 explains key techniques used, including a language for modeling stakeholder requirements, model-driven software development and model-driven transformations; Section 3 focuses on enforcing a role-based access control pattern, and reports several applications of the tool. Section 4 discusses related work and Section 5 highlights several interesting directions in our future work.

[1]https://se.cs.toronto.edu/trac/ome

2. BACKGROUND

In this section, we first use a metamodel for representing stakeholder requirements, then explain necessary concepts of model transformation.

2.1 Stakeholder requirements models

As a modeling language, the distributed intentions (i*) language [13] and its variants support early analysis of stakeholder requirements. These i* models serve as a starting point to further specification of software systems, through a systematic methodology and processes, Tropos [14].

The Tropos approach models early stakeholders' requirements as intentions. The strategic dependencies (SD) between stakeholder roles reflect the corresponding dependencies between their intentions. The strategic rationale (SR) of a stakeholder role is represented by refinement relationships among the intentions within the stakeholder role, such as AND/OR decompositions or MAKE (++) /HELP (+) /HURT (−) /BREAK (−−) contributions. When refinements are done thoroughly, intentions at the bottom can be further operationalised into concrete tasks at the design stage.

Intentions of stakeholder roles can be further classified into goals, softgoals, tasks, resources. The stategic dependencies can be further classified into ownership, permission, and delegation, through an extension by Secure i*[6]. In this paper, to illustrate the model transformations concepts, we use a simplified metamodel that can be extended to any sophisticated metamodels with proper conceptual mappings. A security pattern defined in one refined i* language may not be automatically reusable in another sibling language, it is thus better to define them in a modeling language that is as general as possible. On the other hand, if a security pattern requires a particular concept (such as roles in this paper), then we cannot sacrifice the expressiveness for generality. A proper tradeoff of the language issue is however beyond the scope of this paper.

2.2 Model queries and transformations

Simply put, model transformation is a transformation where both the source and the target conform to modeling languages (or metamodels). For example, the source model may be a UML class diagram, whereas the target model may be Java classes. In this case, model transformation is a mechanism for model-driven code generation. For those familiar with XML transformations such as XSLT, model queries are similar to XPath or XQuery, where certain submodels need to be selected for the transformation.

In many other cases, model transformations can be used not only for model-driven software development, for example, to convert a certain XML Schema into another. Even though the notions of model query and transformation are for general models, its application in model-driven software development has pushed its limit to support various different kinds of software artefacts, especially for those specified in OMG metamodels. Through the inception of model-driven query and transformation frameworks for OMG models, such as ATL[2] and QVT [15], one can now define the OMG metamodel transformation declaratively.

In this work, we will basically define a security pattern with two parts. The first half handles detection of security problems, by specifying a matching pattern on basis of the modeling language, thus supported model-based query. The second half handles resolution of the identified security problems, by transforming a submodel in the source into another in the target (both source and target are in the same modeling language), thus supported by model-based transformation. We will present concrete example of such transformation using the security pattern in the context of next section.

[2]http://www.eclipse.org/m2m/atl/

3. ENFORCING A SECURITY PATTERN

In this section, we first explain the concept of role-based access control and the rationale to use security patterns for complex activities; then we show the representation of such a security pattern and its enforcement, illustrated by a published example.

3.1 Role-based access control (RBAC)

RBAC model became a NIST standard and ANSI INCITS 359-2004 [17], which enables one to represent separation of duty (SoD). When deployed, it can enforce that every subject user are not assigned mutually conflicting authorization roles.

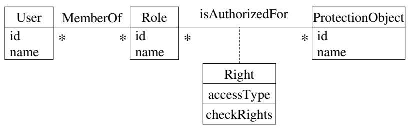

Figure 1: The RBAC pattern by E. Fernandez

Fernandez et al proposed a pattern for RBAC [2] as shown in Figure 1. The pattern corresponds to the core RBAC [17], that is, the access right to objects is not assigned directly to users, but to assigned to roles instead. A user must be a member of a role when he/she wants to access the objects assigned to the role. Thus SoD can be represented as a constraint of `MemberOf` relationship.

The pattern shown in Figure 1 can be easily extended for hierarchical RBAC or for dynamic separation of duty (DSD). For hierarchical extension, one can simply introduce a composite pattern for roles and part-of relationships among users; for DSD, one can introduce a class "session" for representing a temporary assignment of roles to users.

RBAC pattern by E. B. Fernandez enables us to use RBAC in the design phase. However, it is not easy to find suitable and necessary situations to use RBAC in requirement phase: we do not yet have requirements analysis techniques for RBAC application.

Therefore, we explore requirements patterns for RBAC next by using i* goal modeling.

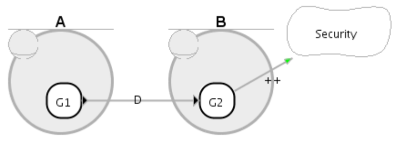

Figure 2: A template of the RBAC pattern in i*

Figure 2 illustrates a simple i* model, in which two stakeholder roles were defined (i.e., *A* and *B*). Given that role *A* delegates to role B a goal G_2 in order to fulfil its own goal G_1, and the goal G_2 in role B is required to fulfil a Security softgoal, denoted by a MAKE contribution). Then one must make sure that G_2 cannot be fulfilled by A. Otherwise, a malicious stakeholder may play the role of A and do harm to the security by performing the goal G_2. As a counter measure, one can add an additional requirement that whenever G_1 is fulfilled, goal G_2 cannot be fulfilled by role A, denoted as a BREAK contribution from G_1 to G_2 inside the strategic rationale of A. Using the augmented goal model, a SAT-solver based reasoning algorithm can detect such malicious case given that it is impossible to satisfy the G_1 and the G_2 by the same role. Figure 3 shows the resulting i* model after the security pattern is applied. It requires a transformation to take into account the matching pattern and create a goal and a contribution link corresponding to the counter measure.

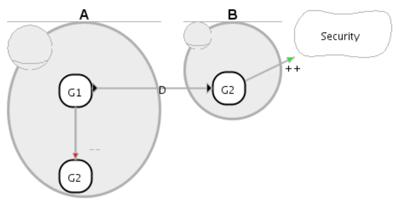

Figure 3: i* model suggested by the security pattern

3.2 Representing a security pattern

In order to detect a security pattern, we must represent the pattern in a structured way.

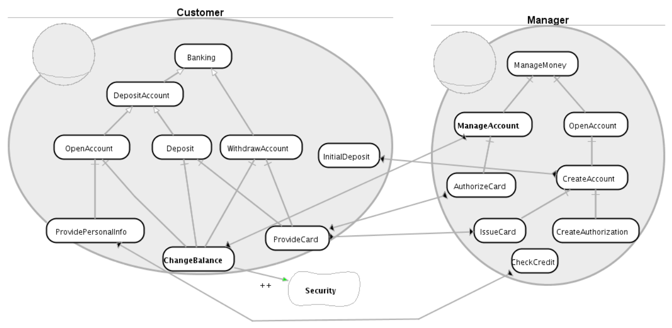

Figure 4: An i* goal model that matches with the security pattern in Figure 2: matched goals are bold.

We first analyse the *context* of the pattern, that is, when does it apply? The following rules form such a context:

- stakeholder A has an intention G1, stakeholder B has an intention G2;
- G1 strategically depends on G2;
- G2 contributes to a Security softgoal through ++ or +;

In banking applications, at least two roles, namely Customer and Manager are involved. Figure 4 shows an example context in the RBAC security pattern we are interested. Here A = Manager, B=Customer, G_1=ManageAccount, G_2 = ChangeBalance. Then a malicious attacker M plays the role of stakeholder A, by sharing the same knowledge in terms of intentions, while though G_2 is depended by G_1, M can carry out G_2 by itself.

Figure 5 shows an example of such malicious attack scenario where RBAC security pattern we are interested. Here A = Manager, B=Customer, M=Attacker, which plays the role of a Manager. Changing the balance on its own, the attacker can fulfil the malicious goal: TransferMoney. As a

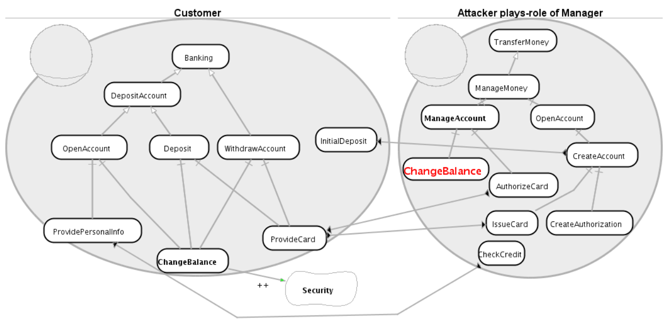

Figure 5: The i* goal model about the attack scenario based on the RBAC security pattern

result, due to the unavailability of permission control, the malicious attacker can issue a card with arbitrary amount. To prevent it from ever happening, a fix is to disallow the attacker to change the balance of the account, either by withdrawing, depositing or initialising a non-negative amount. The prevention must be intentionally carried out through an explicit requirement on stakeholder A such that whenever G1 is satisfied, G2 cannot be satisfied by A itself, see Figure 6.

To sum up, the counter measure leads to the following changes to the model:

- stakeholder A has an intention G_1, stakeholder B has an intention G_2;
- G_1 strategically depends on G_2;
- G_2 contributes to a security softgoal Security through ++ or +;
- A has an intention G_2, and G_1 contributes to G_2 through $--$ relationship.

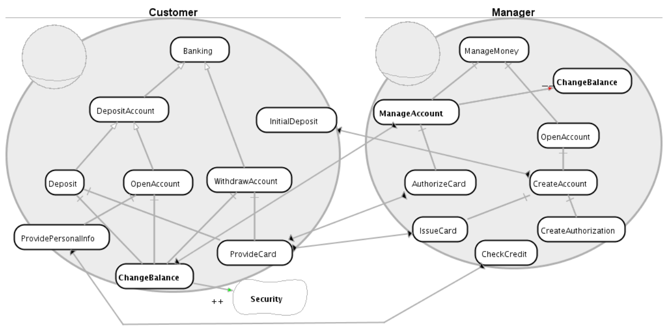

Figure 6: The i* goal model after applying the RBAC security pattern in Figure 3 (cf. Figure 4).

3.3 Enforcing the security pattern

Given that we have informally defined the security pattern, in order to enforce it, we need to describe it formally.

In this work, we use ATL to specify the security pattern as a conditioned model transformation. The condition is boolean, it is true only when the model matches the pattern. In the true case, we perform a non-identical transformation, otherwise, we perform an identical transformation.

```
 1 module RBACSecurityPattern;
 2 create OUT : G from IN: F;
 3 rule Model {
 4   from s:   F!Model
 5   to t:G!Model (name <-s.name,
 6       stakeholders <-s.stakeholders,
 7       intentions <- s.intentions,
 8       dependencies <-s.dependencies,
 9       decompositions <-s.decompositions,
10       contributions<-s.contributions)
11 }
12 rule Stakeholder {
13    from s:F!Stakeholder
14    to    t:G!Stakeholder(name<-s.name,
15           intentions <- s.intentions)
16 }
17 rule Intention {
18    from s:F!Intention
19    to    t:G!Intention(name <- s.name)
20 }
21 rule Decomposition {
22    from s:  F!Decomposition
23    to    t:  G!Decomposition(
24     type <- s.type, -- AND, OR
25     source <- s.source, target <- s.target)
26 }
27 rule Contribution {
28    from s:F!Contribution
29    to    t:G!Contribution(
30          type <- s.type, --MAKE, HELP, HURT, BREAK
31          source <- s.source,
32          target <- s.target)
33 }
34 rule Dependency {
35    from s:F!Dependency(
36         s.dependencyTo.rule->select(e |
              e.target.name = 'Security')->isEmpty()
               or not (s.type ='MAKE' or s.type='
             HELP'))
37    to t:G!Dependency(
38         dependencyFrom <- s.dependencyFrom,
39         dependencyTo    <- s.dependencyTo,
40         dependencyType <- s.dependencyType)
41 }
42 rule Security {
43    from s:  F!Dependency
44          (not s.dependencyTo.rule->select(e |
                e.target.name = 'Security')->isEmpty
                () and (s.type='MAKE' or s.type='HELP
                ') )
45    to    t:  G!Dependency  (
46          dependencyFrom <- s.dependencyFrom,
47          dependencyTo <- s.dependencyTo,
48          dependencyType<-s.dependencyType),
49          t1:G!Contribution(
50                source <-s.dependencyFrom,
51                target <-t2,
52                type <- 'BREAK'),
53          t2:G!Intention(
54                name <- s.dependencyTo.name,
55                stakeholder<-
                       s.dependencyFrom.stakeholder)
56 }
```

An ATL transformation is specified based on a metamodel specified using Eclipse Modeling Framework (EMF)[3]. In earlier work, we have implemented an EMF specification for i* goal models in the OpenOME tool, in which Model, Stakeholder, Intention, Decomposition and Contribution are defined formally as classes of the metamodel. In the ATL specification of the model transformation, we first define identical transformation rules for Model, Stakeholder, Intention, Decomposition, Contribution objects (lines 3-33), because none of them are required to change. Here both the input ("IN") and the output ("OUT") models belong to the same metamodel, respectively represented by F and G in line 2. Every ATL transformation rule has two parts, the "from" part declares a matching model element whilst the "to" part declares a targeting model element. If the identifical transformation rule copies an object, all fields of the object need to be assigned ("←") with the corresponding fields from the object in the input model.

[3]http://www.eclipse.org/modeling/emf/

In case the OCL condition in line 36 is false, the rule for Dependency is none-identical because the target dependency object contributes to a security intention. The condition is true when there is a positive (MAKE or HELP) contribution to a security intention from the target intention of a dependency (i.e., the "dependencyTo" field of the dependency object).

The negation of the condition appears in the Security rule (line 44), which specifies what to change after the enforcing transformation resolve this security problem (lines 45-55). Here a security intention is identified simply by the name "Security", whereas in other sitations the security intention can be defined by a more complex condition. As a result of the transformation (the "to" part), three objects will be created in the output model. The first object (t) is a copy of the input dependency, whilst the other two objects introduce a new BREAK contribution object ($t1$) to a new intention object ($t2$) that is the copy of the target of the original dependency object. Note that the ordering of t, $t1$ and $t2$ does not matter, e.g., $t2$ can be referenced in $t1$.

3.4 Applications

We have directly applied the above ATL query and transformation on the running example, which transform Figure 4 to Figure 6. Then we applied the transformation on 12 examples taken from the OpenOME repository, 4 of them exhibit security concerns [4, 18, 19, 20] and all the three represented in i* language can apply the RBAC security pattern, including Smartcard authentication [18, 19], Secure and trust in healthcare case studies [4]. The high applicability (75%) indicates that perhaps the RBAC security pattern is quite common in i* modeling.

We can find several examples of i* models in literature that can apply our security pattern. Gordijn et al proposed a method for e-Service design using i* and e3value, and they used an example of Internet radio service [21]. In the example, rights society and musician played two roles, namely, "Fee repartitioner" and "Rights holder". Our pattern can be applied to this structure if the repartitioning task is contributing to a security softgoal. If either of them plays both roles at the same time, such a repartition can never be justified. In addition to this example, we have also found several examples in literatures, e.g., technical meeting management [22], reimbursement of medical cost [23] and common electric purse system [24], where our pattern can be applied by adding security related softgoals. According to this survey, we find our pattern can prevent i* modelers from missing security related softgoals because several dependencies in i* models are only partially matching our RBAC pattern.

4. RELATED WORK

Our approach is the first one to formally represent and enforce a security pattern to goal-oriented requirements models by using i* and model-driven transformation. Nonetheless, our approach bares resemblance to several existing approaches including formal representation of security patterns and goal-oriented analysis of non-security patterns.

There are several researches on describing security patterns formally by using some formal language[25, 26]. Schumacher used a symbolic ontology representation language F-Logic to define a knowledge base composed of security ontology, mappings between security concepts and security pattern elements, and inference rules among patterns[25]; although this approach might be useful for retrieving security patterns, there is no mechanism for enforcing those patterns to the target software. Supaporn et al. proposed an another approach to construct extended-BNF-based grammars of security requirements from security patterns in order to translate from security needs of any projects or organizations to requirements[26]. Moreover, they also proposed a prototyping tool for defining security requirements based on the constructed grammars. However, the approach lacks an ability to enforce security patterns to (not newly defined but) existing requirements.

Several approaches analysed software patterns by using goal-oriented (or related) modeling lauguages, such as Non-Functional Requirements Framework (NFR) for any pattern[27] and i* for architecture patterns[28]. These approaches used formal modeling languages for capturing relations between non-functional properties and design decisions, or, relations among architectural elements. On the contrary, we used i* goal models for capturing security properties, before and after applying security patterns.

5. CONCLUSIONS AND FUTURE WORK

In this work, we showed that the ATL model query and transformation framework is suffice to express and enforce a RBAC security pattern in stakeholder requirements models. Such enforcing transformations of security patterns can be seen as operationalizations of security requirement aspects [20]. In future work, we will use bi-directional transformation [29] to synchronize the models before and after enforcement.

Acknowledgements

The authors appreciate the support of the Security in Software Engineering project jointly funded by the NII-GRACE center in Japan and the Royal Society in UK.

6. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley, 1995.

[2] E.B. Fernandez and R. Pan. A pattern language for security models. In *Proc. of Conference on Pattern Languages of Programs (PLoP)*, 2001.

[3] I. Sommerville. *Software Engineering.* Addison-Wesley, 2006.

[4] Lin Liu, Eric Yu, and John Mylopoulos. Security and privacy requirements analysis within a social setting. In *Proc. of International Conference on Requirements Engineering (RE)*, pages 151–161, 2003.

[5] Charles Haley, Robin Laney, Jonathan Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Softw. Eng. (TSE)*, 34(1):133–153, 2008.

[6] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. In *Proc. of RE*, volume 5, 2005.

[7] CB Haley, RC Laney, JD Moffett, and B. Nuseibeh. The effect of trust assumptions on the elaboration of security requirements. In *Proc. of RE*, pages 102–111, 2004.

[8] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of International Conference on Software Engineering (ICSE)*, pages 148–157, 2004.

[9] G. Sindre and A.L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

[10] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett. Introducing abuse frames for analysing security requirements. In *Proc. of RE*, pages 371–372, 2003.

[11] F. Massacci, M. Prest, and N. Zannone. Using a security requirements engineering methodology in practice: The compliance with the Italian data protection legislation. *Computer Standards & Interfaces*, 27(5):445–455, 2005.

[12] Y. Asnar, P. Giorgini, F. Massacci, A. Saidane, R. Bonato, V. Meduri, and C. Riccucci. Secure and Dependable Patterns in Organizations: An Empirical Approach. In *Proc. of RE*, pages 287–292, 2007.

[13] E.S.K. Yu. *Modelling strategic relationships for process reengineering.* PhD thesis, University of Toronto Toronto, Ont., Canada, Canada, 1996.

[14] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[15] Bas Graaf, Sven Weber, and Arie van Deursen. Model-driven migration of supervisory machine control architectures. *J. Syst. Softw.*, 81(4):517–535, 2008.

[16] F. Budinsky, S.A. Brodsky, and E. Merks. *Eclipse Modeling Framework.* Pearson Education, 2003.

[17] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224 – 274, Aug. 2001.

[18] E. Yu and L. Liu. Modelling Trust for System Design Using the i* Strategic Actors Framework. In *Trust in Cyber-Societies-Integrating the Human and Artificial Perspectives*, pages 175–194, 2001.

[19] L. Liu, E. Yu, and J. Mylopoulos. Security Design Based on Social Modeling. pages 71–78, 2006.

[20] Yijun Yu, Julio Cesar Sampaio do Prado Leite, and John Mylopoulos. From goals to aspects: Discovering aspects from goal models. In *Proc. of RE*, pages 38–47, 2004.

[21] Jaap Gordijn, Eric Yu, and Bas van der Raadt. e-service design using i* and e3value modeling. *IEEE Software*, 2006.

[22] Hugo Estrada et al. An empirical evaluation of the i* framework in a model-based software generation environment. In *Proc. of CAiSE*, pages 513–527, 2006.

[23] Volha Bryl, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Designing security requirements models through planning. In *Proc. of CAiSE*, 2006.

[24] Haralambos Mouratidis, Jan Jurjens, and Jorge Fox. Towards a comprehensive framework for secure systems development. In *Proc. of CAiSE*, 2006.

[25] Markus Schumacher. *Security Engineering with Patterns: Origins, Theoretical Model, and New Applications.* LNCS, Vol.2754, Springer, 2003.

[26] Thongchai Rojkangsadan Kawin Supaporn, Nakornthip Prompoon. An Approach: Constructing the Grammar from Security Pattern. In *Proc. of International Joint Conference on Computer Science and Software Engineering (JCSSE2007)*, 2007.

[27] Ivan Araujo and Michael Weiss. Linking Patterns and Non-Functional Requirements. In *Proc. of PLoP*, 2002.

[28] Xavier Franch Gemma Grau. A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures. In *Proc. of European Conference on Software Architecture (ECSA)*, 2007.

[29] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *Proc. of IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 164–173, 2007.