C-SCRIPT: Collaborative Security Pattern Integration Process

Rahma Bouaziz, Fatma Krichen, Bernard Coulette

C-SCRIP:
Collaborative Security Pattern Integration Process

Rahma Bouaziz, Department of Computer Science, Taibah University, Al-Madina Al-Munawarah, Saudi Arabia & ReDCAD, University of Sfax, Sfax, Tunisia

Fatma Krichen, ReDCAD, University of Sfax, Sfax, Tunisia

Bernard Coulette, IRIT, University of Toulouse, Toulouse, France

ABSTRACT

Collaboration is the act of working together, towards a common goal. Collaboration is essential to the success of construction project. In software engineering projects, understanding and supporting collaboration gives the broad impact on product quality. There appears that it is difficult to effectively interact and achieve a common project goals within the bounds of cost, quality and time. The purpose of the paper is to propose a collaborative engineering process, called Collaborative SeCurity patteRn Integration Process (C-SCRIP), and a tool that supports the full life-cycle of the development of a secure system from modeling to code.

Keywords: *CMSPEM, Component Based Systems, Collaborative Process, Security Patterns*

INTRODUCTION

Security pattern are considered as a good solution proposed by security experts to solve a recurrent problem in a given context. However, along with increasing popularity of patterns for security engineering, there is a need for directives and guidelines helping system designers – who are generally not security experts – to implement secure software systems based on set of security patterns. So far there is no clear, well-documented and accepted process dealing with the full integration of security patterns from the earliest phases of software development until the generation of the application code (Devanbu & Stubblebine, 2000).

Our work investigates how non-security experts can take profits from security patterns to easily implement secure component-based applications. In previous work (Bouaziz, Kallel, & Coulette, 2013) (Bouaziz & Coulette, 2012), we proposed an engineering process, called SCRIP (SeCurity patteRn Integration Process), which provides guidelines for integrating security patterns

into component-based models. SCRIP defines activities and products to integrate security patterns in the whole development process, from UML component modeling until aspect code generation.

In this paper, we put the emphasis on the collaborative aspect of the proposed process. We use an extension of the SPEM standard − called CMSPEM − that was introduced in (Kedji, Coulette, Nassar, & Racaru, 2014). We aim to present how software engineers can collaborate to model and implement secure distributed applications.

We propose MDA-based approach whose main interest is to design applications by separating concerns and placing the concepts of models, meta-models and model transformations at the very center of the development process. Our approach combines model-to-model transformation and aspect-oriented programming. In the modeling phase, the designer model his application using UML 2 and take advantages of UML profiles and ATL as model-to-model transformation language to automatically integrate the security patterns in component-based applications. The use of aspect-oriented programming in the implementation phase guarantees the application of the security patterns independently of any application domain.

We build upon an integration process to help designers apply security pattern's solutions in practical situations and to work with patterns throughout a component based software lifecycle (Bouaziz & Coulette, 2012). This process is highly collaborative, since it involves several types of participants who must work together in a coordinated manner. In order to provide a clearer comprehension of the phases of the method, a CMSPEM specification of the proposed process has been produced.

The paper is structured as follows. In the next section we present a background of the work. In Section 2, we present motivations. In Section 3, a collaborative SPEM process for security pattern integration is presented. Section 4 shows detailed description of the proposed collaborative process. A tool prototype SCR-Tool is presented in Section 5. In Section 6, we detail the related work and we conclude the paper in last section.

BACKGROUND

New technologies have emerged during the last decade, such as patterns, model driven engineering and component-based approach. All these technologies have the same common objective that is to facilitate the construction and understanding of software systems. They operate at different levels of abstraction from the general architecture of the system until its implementation. We give below a brief overview of these three technologies.

Component Based Software Engineering

CBSE (Szyperski, 2002) allows building large systems by assembling reusable components. It is a good solution to optimize the time and cost of software design while still guaranteeing the quality of the software (Brown & Wallnau, 1998). Usually, a component is seen as a black box that provides and requires services through its interfaces. Modeling component-based applications consists in describing components, their required and offered services and then describes component instances and finally how these instances are connected to form the final system. At the specification level a system is described as a static interconnection of software components. At runtime a component assembly is an instantiation of an architecture composed of linked component instances.

Model Driven Engineering

A model is a representation of a system or part of system using a well-defined language such as UML (Unified Modeling Language) (OMG, 2010b). Model Driven Engineering (MDE) promotes the use of models at different levels of abstraction and during a system development lifecycle. Compared with approaches traditionally used in which the code represents the key artifact and models are used only for documentation purpose, MDE aims at using models to automatically produce the code. In other words, the developer can obtain the source code by making a series of model transformations to obtaining the final implementation of a system. Model-Driven Architecture (MDA) framework (Bézivin & Gerbé, 2001) (OMG, 2010a) is a model driven approach defined by the Object Management Group (OMG) (OMG, 2011) which allows to put MDE in practice through a set of standards such as UML and UML profiles that extend UML to target a more specific problem domain. A profile can be seen as a set of stereotypes definition and constraints. A stereotype is a mechanism to create a new domain specific model element from an existing one.

Design Security Patterns

Software pattern is derived from the notion of pattern defined by (Alexander, Ishikawa, & Silverstein, 1977) in the building architecture context. In (Alexander et al., 1977) the author suggests that the proposed solution of design should be documented and reused for future designs and he calls them 'patterns'. A pattern is described in terms of context, problem, detailed solution and consequences. Authors of (Yoder & Barcalow, 1998) are pioneers in the field of security patterns. Then several papers have been proposed to model security mechanisms based on the object oriented approach (Ebmayr, Pernul, & Tjoa, 1997). Security patterns accumulate knowledge about security solutions in a structured way. Security patterns are presented to provide guidelines for secure system development. As they present a generic model for security solution mechanism they are also useful to understand complex systems and to teach security concepts independently of any application context.

Aspect Oriented Programming

The principal characteristic of Aspect Oriented Programming (AOP) (Kiczales et al., 1997) is the separation of concerns into two categories: (i) functional concerns that present business code and (ii) technical concerns (aspects) that correspond to nonfunctional requirements. The separations of different types of concerns improve the modularity of applications. An aspect is formed by a point-cut and an advice code. A point-cut may involve one or more aspects. A point-cut is composed of one or many join-points. A join-point can capture specific events where an aspect can be weaved. An advice code is a mechanism, similar to a method, used to codify the code to execute in all join-points of the corresponding point-cut. The advice code can be executed before, after or around a join-point.

To get the application that integrate functional and technical concerns an aspect weaver is used. Two types of weaver exist: static weaver ensures weaving before starting the execution of the application, and dynamic weaver guarantees the weaving at runtime (during the execution of the application).

Software and System Process Engineering Metamodel (SPEM)

In this work we will use SPEM (OMG, n.d.) as software process modeling OMG standard. SPEM meta-model allows describing software development processes. Its purpose is also to allow processes reuse and documentation. It is structured as both a meta-model conforms to MOF and a UML profile.

Hereafter, SPEM terminology is used to specify the phases, roles and steps that are used to describe the SCRIP process. One of the most important principles of SPEM 2.0 is the distinction between *MethodContent* elements (mainly *TaskDefinition*, *RoleDefinition* and *WorkProduct-Definition*) and Process Space (mainly Activity, *TaskUse*, *RoleUse*, *WorkProductUse*). More precisely, method content elements are generic reusable elements described via the package *MethodConent*, whereas process elements reuse them via packages *ProcessWithMethods* and *ProcessStructure*. For example, several instances of *TaskUse* may reuse the same instance of *TaskDefinition*. Concretely, a Process is composed of Activities, which can contain other Activity instances and *MethodContentUse* elements (*TaskUses*, *RoleUses*, *WorkProductUses*).

In the following, we present the SPEM meta-classes that are used to describe our process. *TaskUse* describes a piece of work performed by one *RoleUse*. Tasks can be divided into steps that describe subunits of work needed to perform the task. *RoleUse* defines responsibilities over specific *WorkProducts*, which are consumed/produced in specific activities. *WorkProductUse* (generally called artifact) is anything (piece of information, document, model, source code, etc.) produced, consumed, or modified by a process.

We also reuse the concepts of Phase and Lifecycle – inherited from SPEM 1−which are now defined in the SPEM 2.0 plug-in. A Phase is a specialization of *WorkDefinition* such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria.

We use CMSPEM (Kedji et al., 2014), a metamodel for the description of collaborative software processes and an extension of the SPEM standard, to describe the collaborative aspect of our process. CMSPEM introduces new concepts to represent collaborative processes, and relationships among them. For describing collaborative activities, CMSPEM introduces the concept of Actor (human actor), a specific human participant in a project, associated with a role and provides relations to specify what is done by each actor. CMSPEM also introduces the concept of *ActorSpecificWork*, which is a specific unit of work done by an Actor in the context of a task (*TaskUse* in SPEM), and the concept of *ActorSpecificArtifact*, which is the personal copy of a product (*WorkProductUse* in SPEM), in the workspace of a given Actor.

MOTIVATIONS

Most of the attacks on software systems are based on vulnerabilities caused by software that has been poorly designed and developed (Halkidis, S. T., Tsantalis, N., Chatzigeorgiou, A., & Stephanides, 2008). That's the reason why systems engineers need proven and generic security expert solutions that can be applied to security problems in order to be able to reduce the number of successful attacks against these systems. Security patterns area convenient way of satisfying this need.

Applying security patterns for developing secure software systems is currently a very active area of research (Schumacher, 2003). However, some limitations remain:

First, most of existing approaches as described by (Mouratidis, Giorgini, & Schumacher, 2003) (Fuchs, Gürgens, & Rudolph, 2009) focus on the definition and the application of security patterns in design level without providing any mechanism for implementing these patterns. Con-

versely, some approaches (Horvath & Dörges, 2008) (Diego, Antonio, & Yagüe, 2004) propose concrete implementation of these patterns by providing middleware services that ensure the pattern functionalities. There is little work concerning the full integration of security patterns from the earliest phases of software development, and providing automatic generation of the secure application code (Diego et al., 2004).

Second we note the absence of a comprehensive methodology that assists system developers (non-security experts) when integrating security patterns. There is no guidance on how such security patterns can be integrated into current software component or model based system development methods.

Third, the code that applies security patterns is generally not well modularized, as it is tangled with the code implementing each component's core functionality and scattered across the implementation of different components.

Finally, we note the absence of a process that allows security patterns integration in a collaborative way that promotes working together, towards a common goal.

To overcome these limitations, several works have been done (Ortiz, R., Moral-García, S., Moral-Rubio, S., Vela, B., Garzás, J., & Fernández-Medina, 2010) (Fernandez, Larrondo-Petrie, Sorgente, & VanHilst, 2007). However, all of them were not interested in the collaborative aspect. So in this work, we propose an extension of SCRIP process presented in (Bouaziz et al., 2013) to support collaborative tasks in order to encourage developers to take advantages from security solutions proposed as security patterns in a collaborative way. That is why, in the following, we put the emphasis on the collaborative aspect of the process.

OVERVIEW OF C-SCRIP

Our development process is iterative and incremental: activities are repeated through successive refinements, which allow the reuse of proposed security patterns available in the repository. The structure of our process follows the classical life cycle, in which we have an elicitation phase, a modeling phase and finally an implementation phase.

In the elicitation phase, the designer identifies and models the basic functionality of the system. Security concepts are not introduced.

The modeling phase consists first in identifying and analyzing the security requirements from the application component model. Those security requirements define which security policies are necessary for the analysis model. After that, security patterns are selected to enforce security policies and UML profiles are defined according to the selected security patterns. These patterns are integrated into the application component model in order to obtain a secure Application Component Model.

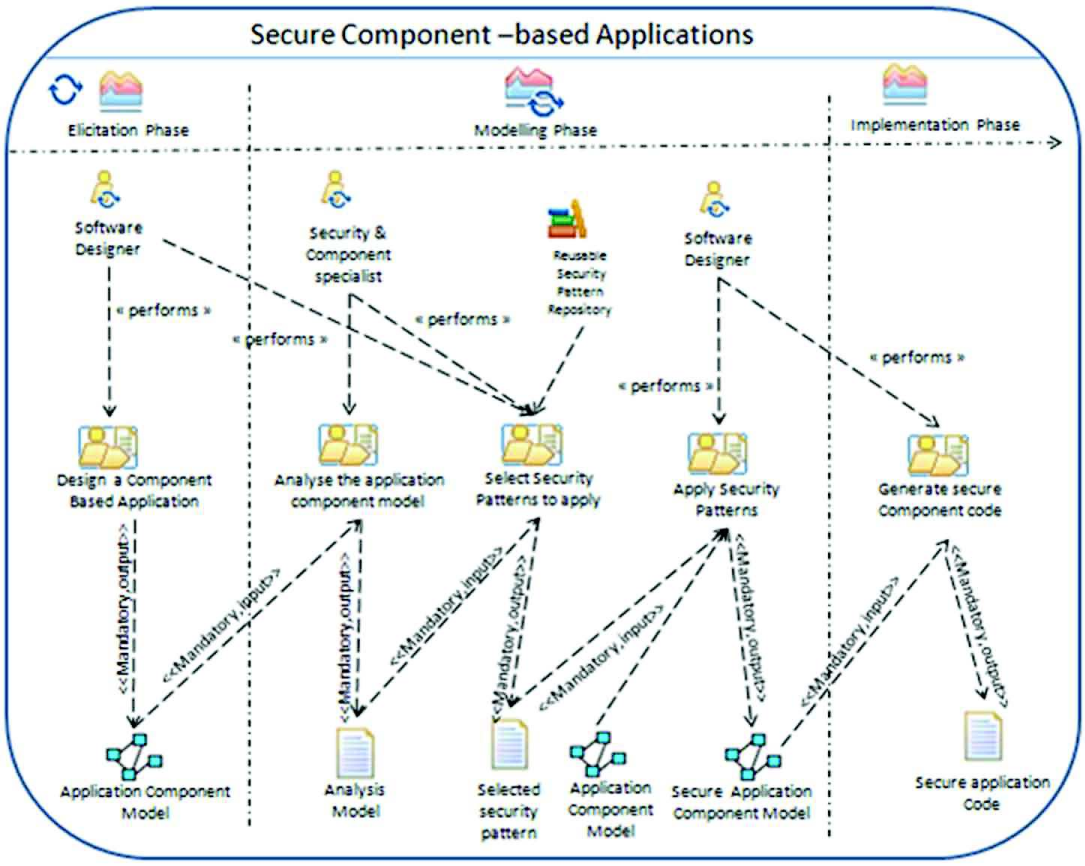
In the implementation phase, a component-based platform must be selected (CCM, EJB, etc.) and the secure application component model is refined into security aspects code together with the functional code for producing the secure application code.

As one can note with this phase, some activities are collaborative, in the sense that several participants working together towards a common goal should perform them. In the following, we put the focus on the collaborative aspects of this phase.

DETAILED DESCRIPTION OF C-SCRIP IN CMSPEM

In this section, we detail our proposed process for security patterns integration in component-based applications. We initially defined our proposed process using SPEM (Software & Systems Process

Figure 1. C-SCRIP - A SPEM process for developing secure component-based applications (one iteration)



Engineering Metamodel) (SPEM, 2015)as described in (Bouaziz et al., 2013) and shown in figure 1. We adopted a concrete syntax with icons partially coming from the SPEM2.0 base Plug-in.

Elicitation Phase

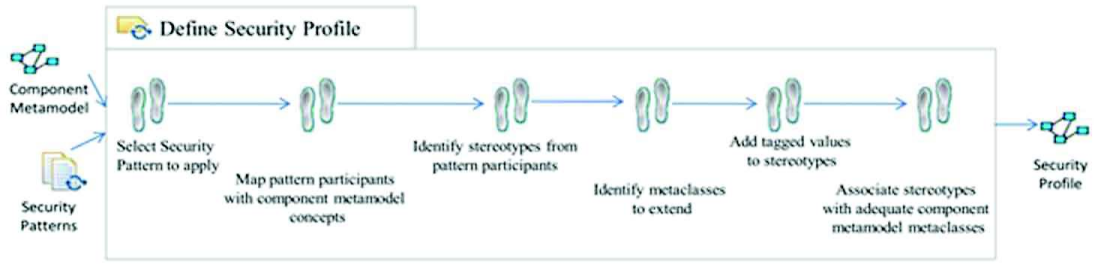
This phase includes one activity "Design Component Based Application", which allows specifying the main functionality of the application. The designer may use the Papyrus suite tool (Papyrus, 2015), for example, to specify his application using UML2 component diagram. He may also use any UML profile that supports specific component models like CCM, EJB or Fractal. The resulting component model does not support any security concept.

Modeling Phase

This phase includes three activities. The first one is the "analysis" activity, which is centered on capturing the requirements of the modeled application. A security repository in which several structures and descriptions of security patterns are stored supports this activity. As shown by the SPEM 2.0 diagram in figure 1, this activity has one mandatory input (Application Component Model).

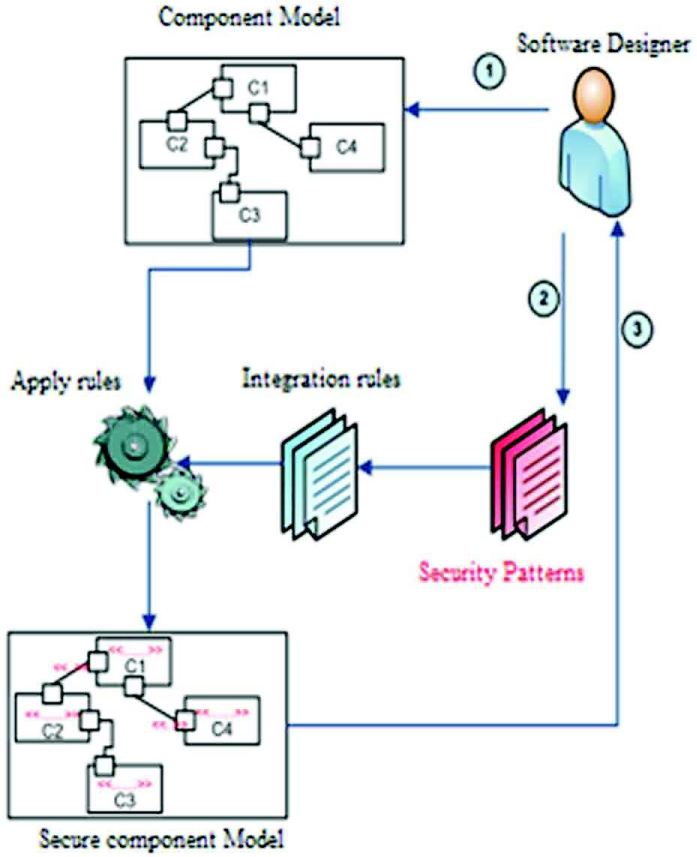
The second activity is the "Select Security patterns to apply" activity in which the designer selects a security pattern from the security pattern repository according to the security require-

Figure 2. Description of the TaskUse: Define Security Profile using SPEM 2



ments and specific application constraint (the analysis model). The designer can select several patterns in an iterative way so as to meet several security requirements to be satisfied in the component-based application.

Figure 3. Procedure for the application of the security patterns integration rules



The third activity is the "Apply security patterns" activity, in which, selected security patterns can be applied to produce a security application component model according to a security policy to be applied.

The application of security patterns is accomplished through the definition of a security profile. This task consists on a match between concepts from security patterns with the concepts of meta-model components. In Figure 2, we present the steps in this TaskUse using the representation proposed in SPEM2.0 standard.

After the definition of the security profile, to semi automate the application of these security profiles. In this perspective, a set of integration rules have been defined. The application of these rules is iterative and related to the choice of security patterns to apply.

The software designer can apply multiple security patterns corresponding to the same security policy.

The following figure 3 shows the steps in the application of these rules.

In the following, we put the emphasis on the implementation phase by showing how it can be described as a collaborative activity.

Implementation Phase

In the rest of this paper, we put the emphasis on this phase, which is dedicated to the production of functional application code and security code (see figure 3). This phase includes the elaboration of two intermediate artifacts: the «Application Functional code» of the component based application and the «Aspect code». «Security specialists» and «Software designers» cooperate to define the final secure application code as explained below.

The «Weaver» (here a software tool) takes application functional code and aspect code as input and delivers a secure code of the application. In this phase, we identified two collaborative activities, as shown by specific icons in figure 3: (1) Produce the aspect code and (2) Produce secure application source code. We identified certain roles that take part in the implementing activity of this process.

Software Designer is responsible for the design of the component-based application and for supporting the definition of security requirements. This stakeholder should contribute with all security aspects for component application. He should collaborate and agree with the remaining stakeholder in this activity in order to produce secure code of the application.

Security specialist leads and coordinates security requirements and integrates them with the system requirements. In particular in this phase, this stakeholder is responsible for the generation of the aspect code according to the secure application model.

Indeed several approaches and commercial tools support the generation of code skeleton with different technologies (EJB, .NET, C++, etc.) from a UML component diagram, based on a set of predefined libraries. The designer can also produce the corresponding code by using for instance the MDA approach. He first transforms the application component model into a platform specific model. The corresponding code is then produced using a model-to-text generator. In our case we used the EJB UML profile for generating functional application code targeting the EJB platform.

We detail artifacts of the "Produce aspect code" activity. The output artifact of this activity is the secure application code model, which is composed of two artifacts produced and used in this activity: Application functional code and Aspect code.

For detailing collaborative activities, in this case, "Produce aspect code" activity for example, we use the notation proposed by (Kedji et al., 2014). In this work, the authors introduce concepts needed to represent precise and dynamic collaboration and propose an extension of the SPEM

Figure 4. Detailed implementation phase

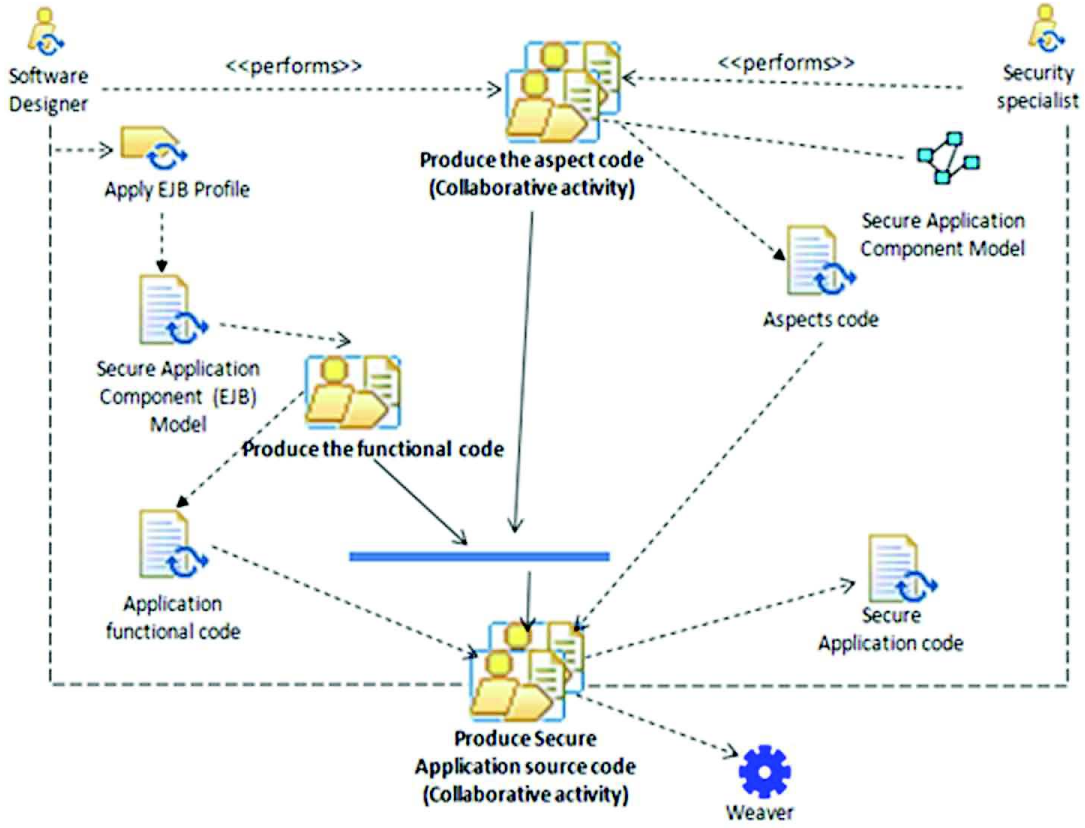


Figure 5. Relations between actors and their specific tasks during the production of aspect code

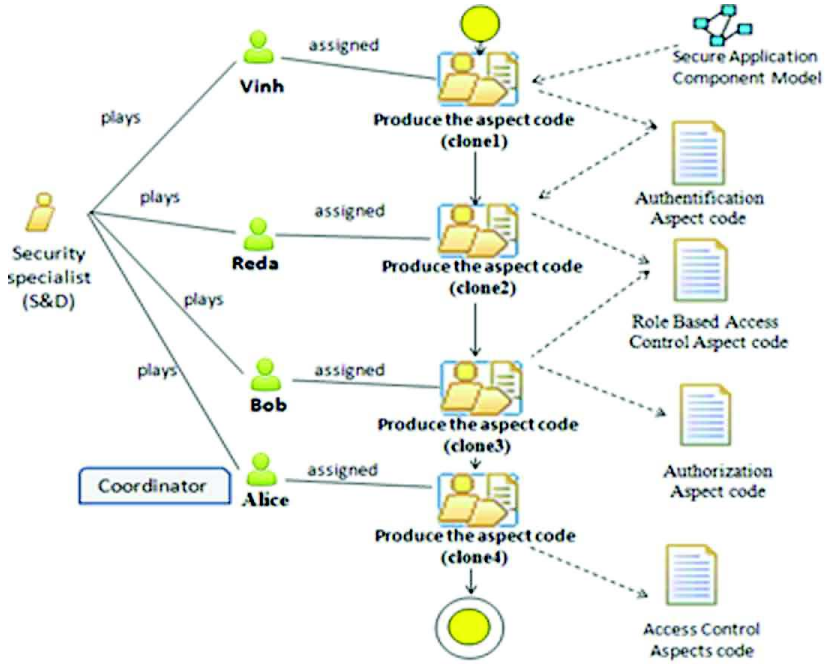
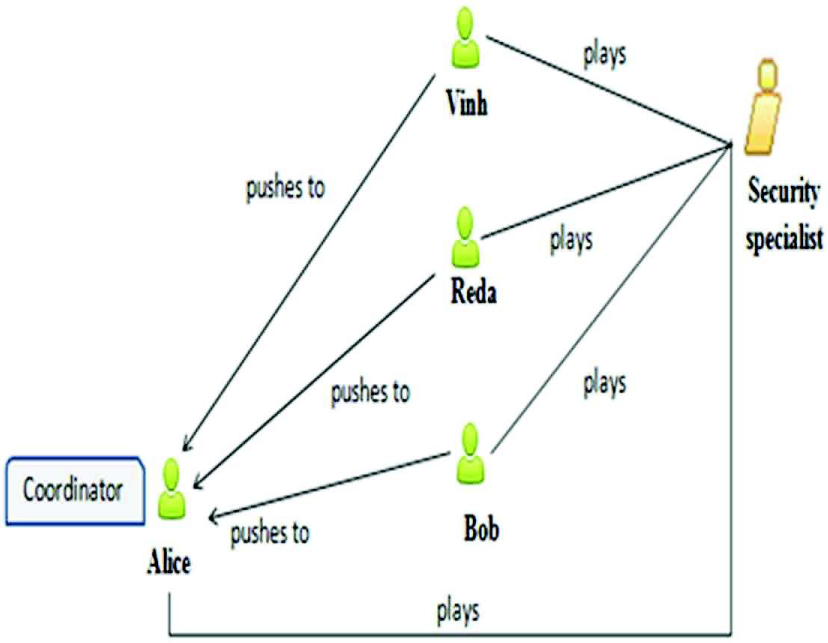


Figure 6. CMSPEM relations between actors



standard by adding the concept of Actor (human actor) ([INSERT FIGURE 001]) associated with a role and adding relations to specify what is done by each actor, products he is responsible for, relations with other actors, knowing that:

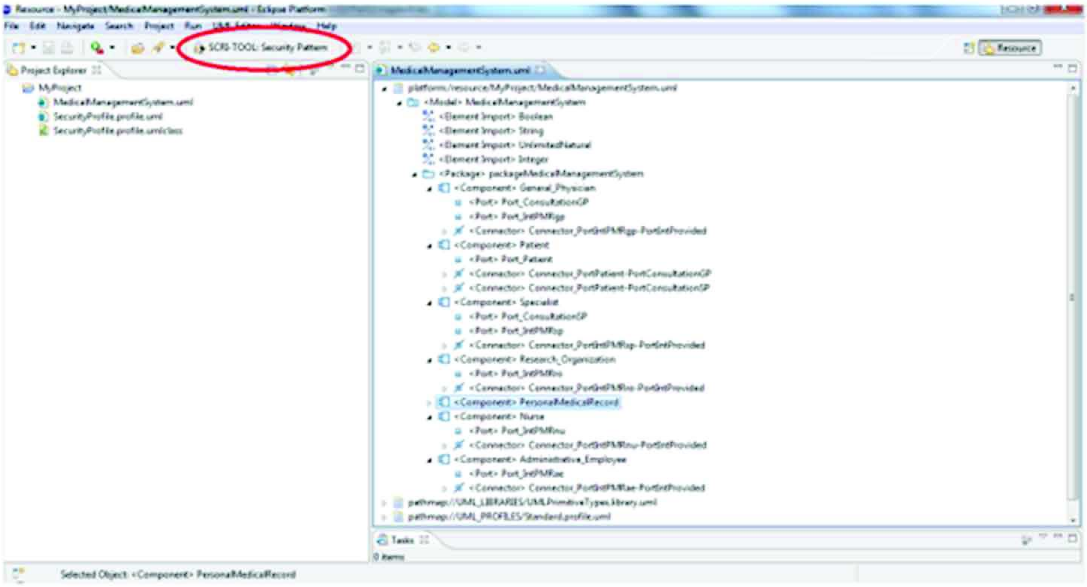
- Each actor plays one or multiple roles.
- Each actor is assigned to one or several activities.
- Each actor owns one or several specific artifacts.

In "Produce the aspect code" activity, as shown in figure 4, we have identified relations between actors and their role, and between actors and their specific tasks.

As mentioned above (section III), to apply one security policy we have to use several security patterns together. For example, to apply the Access control policy, three security-patterns are used: authentication, authorization and role based access control. This activity is qualified as collaborative because the generation of aspects code corresponding to each pattern is assigned to a set of actors. In our collaborative context, each actor enacts the same activity meaning that he works on a copy of the "Produce the aspect code" activity, in a sequential manner. A scenario of collaboration is illustrated in figure 3; for example, "Vinh" is in charge of producing the aspect code related to the "Authentification" pattern; "Reda" is in charge of producing the aspect code related to the "Role based access control" pattern; "Alice" is the coordinator and thus is responsible for generating the global aspect code related to the access control policy.

"Alice" is a Security Specialist, who coordinates the collaborative activity. Each actor (Vinh, Reda, Bob) sends the artifacts he has produced to Alice, like it is explicitly shown in figure 5 with the relation "pushesTo". This type of collaboration can be typically implemented with a versioning management system such as "svn" or "git".

Figure 7.SCRI-TOOL screenshot- Added button to allow security pattern application

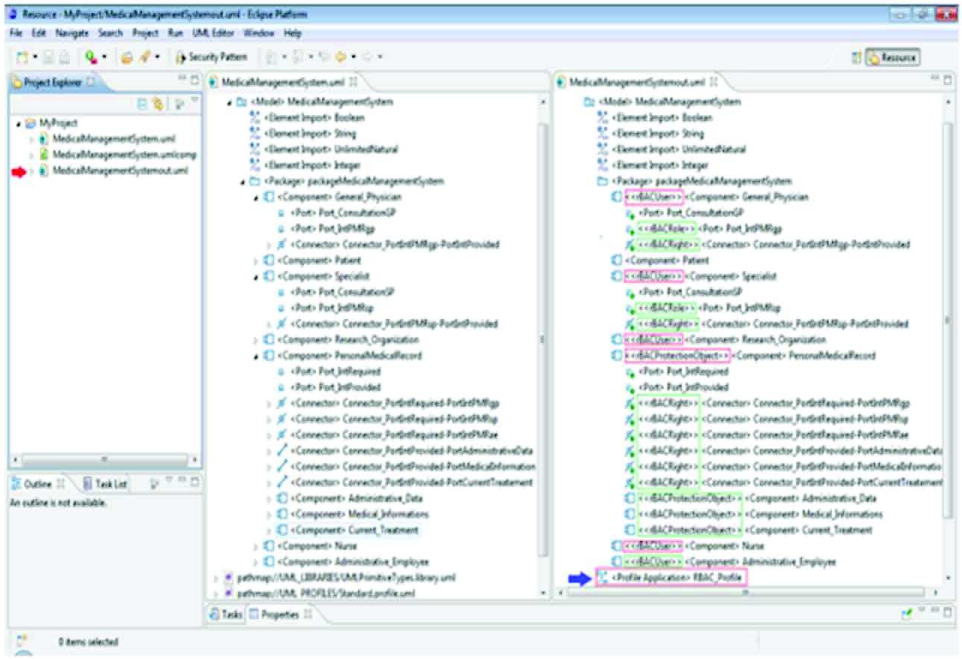


CASE TOOL PROTOTYPE “SCRI-TOOL”

We have developed a case tool prototype based on the Eclipse development platform (see figure 7). So far, this tool is a proof of concept of our collaborative engineering process. We have em-

Figure 8. SCRI-TOOL screenshot-Security policy choice Interface

Figure 9. SCRI-TOOL screenshot-Application of the security pattern



ployed several plug-ins implementing the MDA standard: for instance, the "model development tools"(MDT) for supporting UML and UML profiles. Also we use ATL to specify the pattern integration rules to transform the application model into a secure application model. To design model-to-code transformation we have used Acceleo mappings to automatically implement the final secure application. We have combined the aforementioned defined plug-ins to provide an "integrated development environment" (IDE) named SCRI-TOOL to design secure component based application based on the collaborative engineering process proposed in this paper.

Figure 7 shows a screenshot of the prototype. On the left-hand side of the figure, the illustrative project has been initiated; it proposes to apply a security pattern within the Eclipse Workspace (en circled button). If we create the applicative example, producing a diagram by using the UML model editor from Eclipse, an example related to the management of the medical system is created, Interfaces like in Figure 8 and Figure 9 allow respectively choosing security policy to apply and produce a secure model after pattern application.

The classical menu bar from Eclipse has been adapted to support the code generation options. After generating the Java and AspectJ code we use AJDT (AspectJ Development Tools) for aspects weaving.

Our secure collaborative process contributes to automate the development of secure applications using security patterns solutions. Nevertheless, our proposal has some limitations or constraints:

- The step related to the integration of patterns requires a manual contribution in order to determine which artifact will need security in the application case study.
- The prototype CASE tool, which supports our process, needs to be completed and validated on real projects.
- Our process is only based on direct engineering methods. Developing methods in order to offer direct and reverse engineering methods could enrich the proposal.

- In our approach, so far, we address security based on access control to guarantee confidentiality. However, other security aspects, such as integrity, reliability and availability could be taken into account.

Other kinds of non-functional requirements such as cost-benefit and performance are not included within our process. Our contribution to the technological framework consists mainly of integrating different technologies from the Eclipse Modeling Project and we aim to extending them to obtain collaborative functionality.

RELATED WORK

There are a large amount of works addressing the topic of security design patterns applicability and usability. (Ortiz, R., Moral-García, S., Moral-Rubio, S., Vela, B., Garzás, J., & Fernández-Medina, 2010) provide an analysis of the main works related to security patterns. They discuss their applicability for the analysis and design of secure architectures in real and complex environments. Here, we sum up some of the proposals for integration of security patterns. In (IIndrakshi Ray, Robert France, Na Li, 2004), authors propose a security pattern integration technique dealing with model transformation using ATL. Moreover, authors in (Horvath & Dörges, 2008) use Petri nets to model security patterns at an abstract level. A methodology for integrating security patterns into all stages of the software development lifecycle is proposed in (Fernandez et al., 2007). Other approaches (Georg, Ray, & France, 2002) (Ray, France, Li, & Georg, 2004) present the use of aspect oriented software design approach to model security patterns as aspects and weave them in to the functional model.

Concerning design pattern application, S. Yau (Yau, 2000) uses a formal design pattern representation and a design pattern instantiation technique for automatic generation of component wrappers from design patterns. In addition, several approaches introduce their own tool-based support for pattern instantiation. In (Cinnéide & Nixon, 2001) authors provide an UML profile which allows the explicit representation of design patterns in UML models through a model transformation approach. Authors in (Kajsa & Majtás, 2010) describe an approach for creating automated transformations that can apply a design pattern to an existing program. In (Kajsa & Majtás, 2010), authors propose a method supporting design patterns application in software projects, based on a semantics defined via UML profile and model transformations.

We can conclude that most of existing approaches focus on the application of security patterns at design level without providing any mechanism for implementing them in component-based applications. There is little work concerning the full integration of security patterns from the earliest phases of software development and providing automatic generation of the final secure application code. Even more, the code that applies security patterns is generally not well modularized, as it is tangled with the code implementing each component's core functionality and scattered across the implementation of different components.

To remedy these limitations we have provided a collaborative security pattern integration process –described in SPEM− with tool support in order to encourage developers to take advantage from security solutions proposed in security patterns.

CONCLUSION

In this paper, we have proposed a collaborative engineering process for security pattern integration, by eliciting and developing both functional and security aspects as non-functional requirements. This approach is outlined as follows. First an application model is built, here a component based application model. Second, this model is transformed by using ATL transformations that consist in applying the security profiles stereotypes corresponding to the security policies to enforce. Our process is represented as a result of the application of SPEM, and its extension CMSPEM to represent collaborative aspects of the process. We express collaboration in a formalism well suited for easy representation and tool-provided assistance.

This process has the advantage of separating the application domain expertise and expertise in security. The integration of security in the software development process becomes easier for the architects/designers. Furthermore, it is relatively simple and suitable for use by non-security experts. Understanding security patterns from their description and having knowledge on applications-based components are sufficient skills to use this process.

In this work the implementation and the experimentation presented in the Case Tool Prototype section as a partial validation of our approach because further work is still needed to get a true validation.

As future work, we aim to provide a complete development environment to design secure component based application using the proposed SCRIP engineering process.

Our immediate future work consists of several tasks. Concerning the implementation of our proposal, we have planned to complete the developed tool in order to automatically produce the functional code to target other platforms. In addition, we plan to extend the current version of the prototype to support collaboration aspects so as to clearly show who does what. From a conceptual perspective, we intend to define and implement a decision security patterns map for automatically selecting patterns related to given security policy in a given application.

REFERENCES

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language : Towns, Buildings, Construction*. OUP USA.

Bézivin, J., & Gerbé, O. (2001). Towards a Precise Definition of the OMG/MDA Framework. In *16th IEEE international conference on Automated software engineering* (pp. 273–280). IEEE Computer Society.

Bouaziz, R., & Coulette, B. (2012). Applying Security Patterns for Component Based Applications Using UML Profile. In *IEEE 15th International Conference on Computational Science and Engineering* (pp. 186–193). doi:10.1109/ICCSE.2012.104

Bouaziz, R., Kallel, S., & Coulette, B. (2013). An Engineering Process for Security Patterns Application in Component Based Models. In *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (pp. 231–236). IEEE Comput. Soc. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6570618

Brown, A. W., & Wallnau, K. C. (1998). The current state of CBSE. *IEEE Software*, *15*(5), 37–46. doi:10.1109/52.714622

Cinnéide, M. Ó., & Nixon, P. (2001). Automated software evolution towards design patterns. In *Proceedings of the 4th International Workshop on Principles of Software Evolution* (pp. 162–165).

Devanbu, P. T., & Stubblebine, S. (2000). Software Engineering for Security : a Roadmap. In *Proceedings of the conference of The future of Software engineering*.

io, M., & Yagüe, I. M. (2004). Integration of Security Patterns in Software Models based
criptions. In *UML 2004*. Retrieved from http://ctp.di.fct.unl.pt/UML2004/DocSym/Di-
4DocSym.pdf

ul, G., & Tjoa, A. M. (1997). Access Controls by Object-Oriented Concepts. In *11th IFIP
Conference on Database Security*.

Larrondo-Petrie, M. M., Sorgente, T., & VanHilst, M. (2007). A methodology to develop
sing patterns. In H. Mouratidis & P. Giorgini (Eds.), *Integrating Security and Software
vances and Future Visions* (Vol. 5, pp. 107–126). Idea Group Inc. doi:10.4018/978-1-
05

ıs, S., & Rudolph, C. (2009). Towards a generic process for security pattern integration. In
ernational Workshop on Database and Expert Systems Applications, DEXA* (pp. 171–175).
XA.2009.51

., & France, R. (2002). Using aspects to design a secure system. In *Eighth IEEE Inter-
nce on Engineering of Complex Computer Systems, 2002. Proceedings.* doi:10.1109/
81504

antalis, N., Chatzigeorgiou, A., & Stephanides, G. (2008). Architectural Risk Analysis of
s Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing*,
oi:10.1109/TDSC.2007.70240

ɔrges, T. (2008). From security patterns to implementation using petri nets. In *Proceedings
national workshop* ... (pp. 17–23). New York: ACM Press. doi:10.1145/1370905.1370908

obert France, Na Li, G. G. (2004). An aspect-based approach to modeling access control
ation and Software Technology, (46): 575–587.

ás, L. (2010). Design patterns instantiation based on semantics and model transformations.
in Computer Science (including subseries Lecture Notes in Artificial Intelligence and
Bioinformatics) (Vol. 5901 LNCS, pp. 540–551). doi:10.1007/978-3-642-11266-9_45

llette, B., Nassar, M., & Racaru, F. (2014). Supporting Collaborative Development Using
An Integration-Focused Approach. *Journal of Software: Evolution and Process*, *26*(10),

ıping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., & Irwin, J. (1997). Aspect-
ıming. In M. Aksit & S. Matsuoka (Eds.), *ECOOP* (pp. 220–242). Heidelberg: Springer.

iorgini, P., & Schumacher, M. (2003). Security Patterns for Agent Systems. In *Proceed-
ropean Conference on Pattern Languages of programs*. Retrieved from http://hdl.handle.

*M*DA Specifications. Retrieved Mai 2015, from http://www.omg.org/ mda/specs.htm

Unified Modeling Language (UML). Retrieved Mai 2015, from http://www. uml.org/

ɔject Management Group. Retrieved October 7, 2012, from http://www.omg. org/

García, S., Moral-Rubio, S., Vela, B., Garzás, J., & Fernández-Medina, E. (2010). Ap-
urity patterns. In On the Move to Meaningful Internet Systems: OTM (pp. 672–684).
1007/978-3-642-16934-2_49

Entreprise architect website [online], Retrieved from http://www.papyrusuml.org/ scripts/
ontent/templates /show.asp?L=EN&P=55 &v Ticker=alleza&ITEMID=3.

.., Li, N., & Georg, G. (2004). An aspect-based approach to modeling access control con-
n and Software Technology, *46*(9), 575–587. doi:10.1016/j.infsof.2003.10.007

(2003). *Security Engineering with Patterns: Origins, Theoretical Models, and New Ap-
*ger Berlin / Heidelberg.

SPEM. (2015): Software and systems process engineering metamodel specification (SPEM), version 2.0. Retrieved Mai 2015, from http://www.omg.org/spec/SPEM/2.0/PDF

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. BOSTon, MA, USA: Addison-Wesley Longman Publishing Co.

Yau, S. S. (2000). Integration in component-based software development using design patterns. In *24th Annual International Computer Software and Applications Conference. COMPSAC* (pp. 369–374). doi:10.1109/CMPSAC.2000.884750

Yoder, J., & Barcalow, J. (1998). Architectural patterns for enabling application security. In *Proceedings of PLoP 1997* (Vol. 51, p. 31).