



HAL
open science

A collaborative process for developing secure component-based applications

Rahma Bouaziz, Slim Kallel, Bernard Coulette

► **To cite this version:**

Rahma Bouaziz, Slim Kallel, Bernard Coulette. A collaborative process for developing secure component-based applications. IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises - WETICE 2014, Jun 2014, Parma, Italy. pp. 306-311. hal-01147258

HAL Id: hal-01147258

<https://hal.science/hal-01147258>

Submitted on 30 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : <http://oatao.univ-toulouse.fr/>
Eprints ID : 13023

To link to this article : DOI :10.1109/WETICE.2014.82
URL : <http://dx.doi.org/10.1109/WETICE.2014.82>

To cite this version : Bouaziz, Rahma and Kallel, Slim and Coulette, Bernard *[A collaborative process for developing secure component-based applications](#)*. (2014) In: IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises - WETICE 2014, 23 June 2014 - 25 June 2014 (Parma, Italy).

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

A collaborative process for developing secure component based applications

Rahma Bouaziz^{1,2}, Slim Kallel², Bernard Coulette¹

¹IRIT, University of Toulouse
Toulouse, France
rahma.bouaziz@irit.fr, bernard.coulette@irit.fr

²ReDCAD, University of Sfax,
Sfax, Tunisia
slim.kallel@fsegs.rnu.tn

Abstract—Security patterns describe security solutions that can be used in a particular context for recurring problems in order to solve a security problem in a more structured and reusable way. Patterns in general and Security patterns in particular, have become important concepts in software engineering, and their integration is a widely accepted practice. In this paper, we propose a model-driven methodology for security pattern integration. This methodology consists of 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— Component; Component based systems; Security patterns; Collaborative process; CMSPEM.

I. 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 [1].

Our work investigates how non-security experts can take profits from security patterns to easily implement secure component-based applications. In previous work [11, 21], 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 [10]. We aim to present how software engineers can

collaborate to model and implement secure distributed applications.

Our approach intends to provide a model-driven engineering 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 [21]. 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 motivations of this work. In Section III, a collaborative SPEM process for security pattern integration is presented. Section IV shows detailed description of the proposed collaborative process. A tool prototype SCR-Tool is presented in section V. In section VI, we detail the related work and we conclude the paper in Section VII.

II. MOTIVATIONS

Most of the attacks on software systems are based on vulnerabilities caused by software that has been poorly designed and developed [4]. 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 [5]. However, some limitations remain; in the following we will present some of them.

First, most of existing approaches as described by [6][7] focus on the definition and the application of security patterns in design level without providing any mechanism for implementing these patterns. Conversely, some approaches [8][9] 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 [9].

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.

Also, 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 can 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 [12] [14]. However, all of them were not interested in the collaborative aspect. So in this work, we propose an extension of SCRIP process presented in [11] 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.

III. 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.

To describe the collaborative aspect of our process, we use CMSPEM, an extension of the SPEM standard, proposed by Kedji et al. [10]. 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.

IV. 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 Engineering Metamodel) [3] as described in [11] and shown in figure 1. We adopted a concrete syntax with icons partially coming from the SPEM2.0 base Plug-in.

A. Elicitation phase

This phase includes one activity "Design Component Based Application", which allowsspecifying the main functionality of the application. The designer may use the Papyrus suite tool [2], 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.

B. 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 requirements and specific application constraints

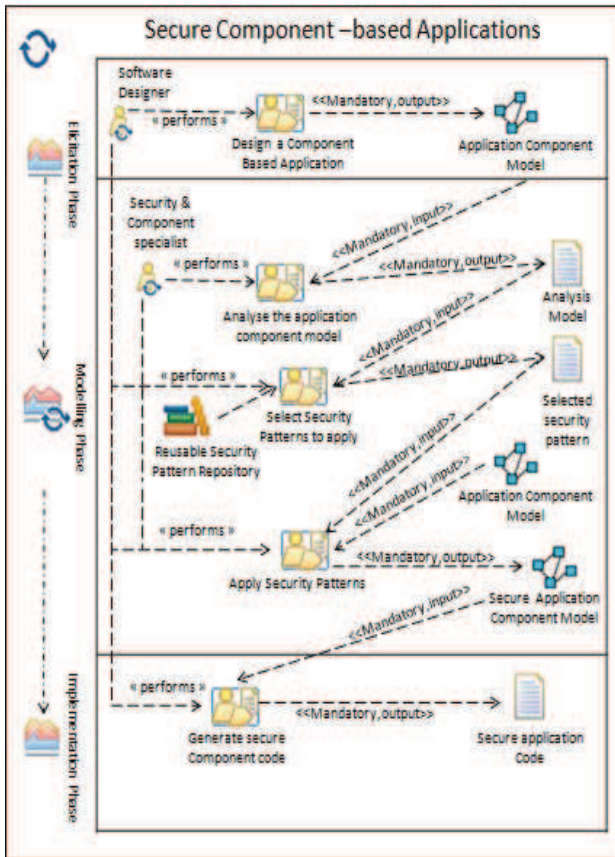


Figure 1. C-SCRIP - A SPEM process for developing secure component-based applications (one iteration)

(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.

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

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

C. 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 2). 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 2: (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.

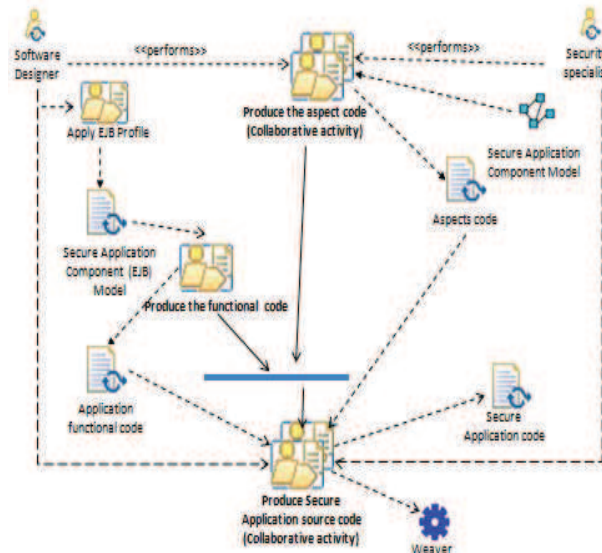


Figure 2. Detailed implementation phase

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.


1) Generating the functional application code

To produce the «Functional code» of the component based application, we reuse existing approaches. 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.

2) Producing the aspect code

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 A.K. Kedji et al.[10]. In this work, the authors introduce concepts needed to represent precise and dynamic collaboration and propose an extension of the SPEM standard by adding the concept of Actor (human actor) () 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 3, we have identified relations between actors and their role, and between actors and their specific tasks.

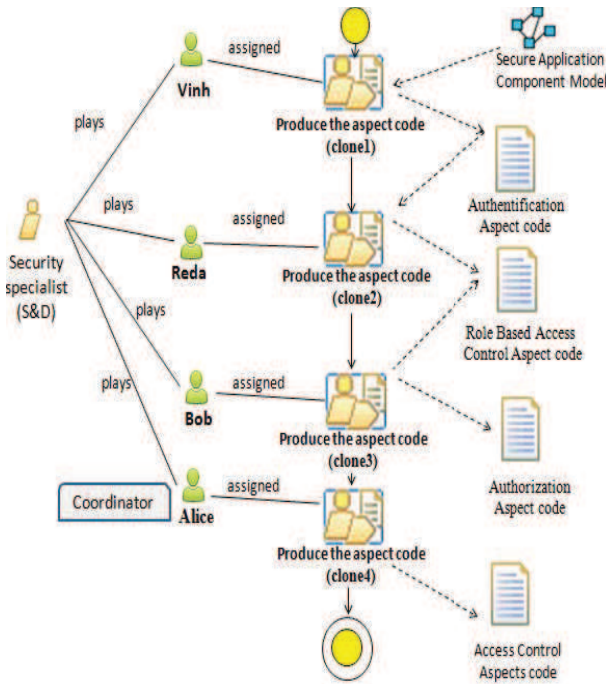


Figure 3. Relations between actors and their specific tasks during the production of aspect code

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 "Authentication" 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 4 with the relation "pushesTo". This type of collaboration can be typically implemented with a versioning management system such as "svn" or "git".

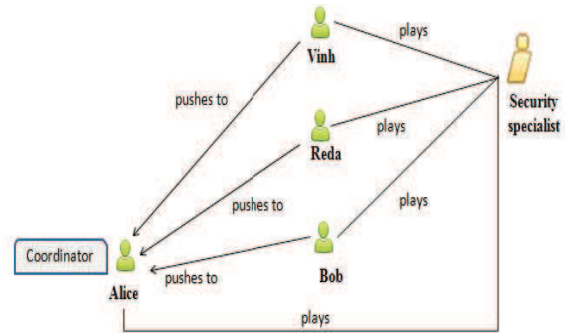


Figure 4. CMSPEM relations between actors

V. CASE TOOL PROTOTYPE "SCRI-TOOL"

We have developed a case tool prototype based on the Eclipse development platform (see figure 5). So far, this tool is a proof of concept of our collaborative engineering process. We have employed 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 Aceleo 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 5 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.

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.

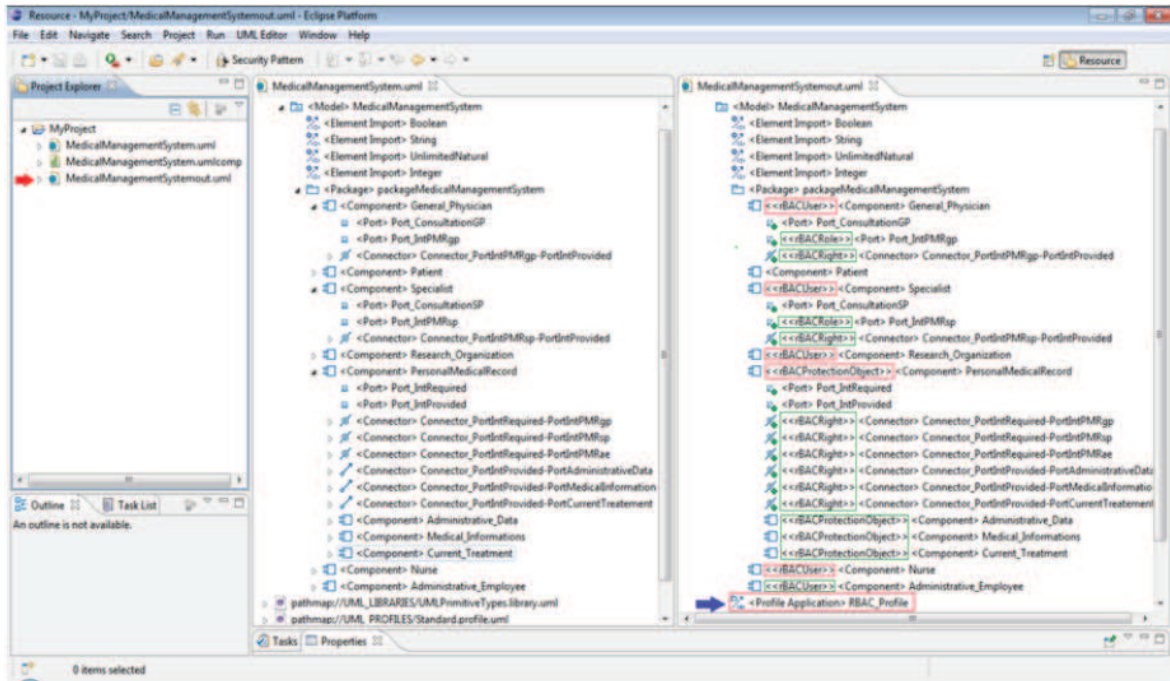


Figure 5. SCRI-TOOL screenshot-Application of the security pattern

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.

VI. RELATED WORK

There are a large amount of works addressing the topic of security design patterns applicability and usability. Ortiz et al. [12] 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 [13], authors propose a security pattern integration technique dealing with model transformation using ATL. Moreover, authors in [8]

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 [14]. Other approaches [15][16] 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 [17] 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 [18] authors provide an UML profile which allows the explicit representation of design patterns in UML models through a model transformation approach. Authors in [19] describe an approach for creating automated transformations that can apply a design pattern to an existing program. In [2], 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.

VII. 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 section Visa partial validation of our approach because further work is still needed to get a true validation.

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

- [1] Premkumar T. Devanbu, Stuart Stubblebine, Software Engineering for Security: a Roadmap in *Proceedings of the conference of The future of Software engineering*, 2000.
- [2] Papyrus UML. <http://www.papyrusuml.org/>
- [3] SPEM 2.0. <http://www.omg.org/spec/SPEM/2.0/>.
- [4] Spyros T. Halkidis, Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, Architectural Risk Analysis of Software Systems Based on Security Patterns, *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 129–142, 2008.
- [5] Markus Schumacher, *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*, Springer-Verlag, 2003.
- [6] Haralambos Mouratidis, Paolo Giorgini, Markus Schumacher, Security Patterns for Agent Systems. Proceedings of the 8th European Conference on Pattern Languages of programs, 2003
- [7] Andreas Fuchs, Sigrid Gürgens, Carsten Rudolph, Towards a Generic Process for Security Pattern Integration. in *Proceedings of the 20th International Workshop on Database and Expert Systems Application*, 2009, pp. 171–175.
- [8] Viktor Horvath, Till Döriges, From security patterns to implementation using petri nets, in *Proceedings of the fourth international workshop on Software engineering for secure systems - SESS*, 2008, pp. 17–24.
- [9] Diego Ray, Antonio Maña, Mariemma I. Yagüe Integration of Security Patterns in Software Models based on Semantic Descriptions
- [10] Komlan Akpédjé Kedji, Redouane Lbath, Bernard Coulette, Mahmoud Nassar, Laurent Baresse, Florin Racaru Supporting collaborative development using process models: a Tooled Integration-focused Approach. *Journal of Software : Evolution and Process (JSEP)*. February 2014, Wiley
- [11] Rahma Bouaziz, Slim Kallel, Bernard Coulette, An Engineering Process for Security Patterns Application in Component Based Models, in *Proceedings of the International conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2013 IEEE Computer Society, pp.231,236, 17-20 June 2013
- [12] Roberto Ortiz, Santiago Moral-García, Santiago Moral-Rubio, Belén Vela, Javier Garzàs, Eduardo Fernández-Medina Applicability of security patterns, in *On the Move to Meaningful Internet Systems: OTM*, 2010, pp. 672–684.
- [13] Yijun Yu, Haruhiko Kaiya, Hironori Washizaki, Yingfei Xiong, Zhenjiang Hu, Nobukazu Yoshioka, Enforcing a security pattern in stakeholder goal models, in *Proceedings of the ACM workshop on Quality of protection*, 2008, pp. 9–14.
- [14] Eduardo B. Fernandez and Maria M Larrondo-Petrie A Methodology to Develop Secure Systems Using Patterns, *Integrating Security and Software Engineering*, vol. 5, pp. 2006.
- [15] Geri Georg, Indrakshi Ray, Robert France, Using Aspects to Design a Secure System, *Proceedings of the Eighth IEEE International Conference on Engineering of Complex Computer Systems*, pp. 117– 126., 2002.
- [16] Indrakshi Ray, Robert France, Na Li, Geri Georg An aspect-based approach to modeling access control concerns, *Information and Software Technology*, vol. 46, pp. 575–587, 2004.
- [17] Stephen S. Yau, Ning Dong, Integration in component-based software development using design patterns, in *Proceedings 24th Annual International Computer Software and Applications Conference*.2000, pp. 369–374.
- [18] Xue-Bin Wang, Quan-Yuan Wu, Huai-Min Wang, Dian-Xi Shi, Research and Implementation of Design Pattern-Oriented Model Transformation, in *Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, 2007, pp. 24–24.
- [19] Mel Ó Cinnéide, Paddy Nixon, Automated software evolution towards design patterns, in *Proceedings of the 4th international workshop on Principles of software evolution*, 2002, p. 162.
- [20] Peter Kajsa, L’ubomír Majtás, Design patterns instantiation based on semantics and model transformations, in *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science*, 2010, pp. 540–551.
- [21] Rahma Bouaziz, Bernard Coulette. Applying Security Patterns for Component Based Applications Using UML profile. In *Proceedings of the International Conference on Computational Science and Engineering*, Paphos, Cyprus, p.186-193, 2012.