International Journal of Electrical and Computer Engineering (IJECE)
Vol.8, No.6, December2018, pp. 5278~5291
ISSN: 2088-8708, DOI: 10.11591/ijece.v8i6.pp5278-5291

A UML Profile for Security and Code Generation

Abdellatif Lasbahani, Mostafa Chhiba, Abdelmoumen Tabyaoui
RMI Laboratory, FSTS Hassan 1st University, Morocco

Article Info

Article history:

Received Jun 23, 2017
Revised Jul 28, 2018
Accepted Aug 11, 2018

Keyword:

Code generation
LOC
MDA
Model transformation
SDSIB
Security profile
Security properties
SPEM

ABSTRACT

Recently, many research studies have suggested the integration of safety engineering at an early stage of modeling and system development using Model-Driven Architecture (MDA). This concept consists in deploying the UML (Unified Modeling Language) standard as aprincipal metamodel for the abstractions of different systems. To our knowledge, most of this work has focused on integrating security requirements after the implementation phase without taking them into account when designing systems. In this work, we focused our efforts on non-functional aspects such as the business logic layer, data flow monitoring, and high-quality service delivery. Practically, we have proposed a new UML profile for security integration and code generation for the Java platform. Therefore, the security properties will be described by a UML profile and the OCL language to verify the requirements of confidentiality, authorization, availability, data integrity, and data encryption. Finally, the source code such as the application security configuration, the method signatures and their bodies, the persistent entities and the security controllers generated from sequence diagram of system's internal behavior after its extension with this profile and applying a set of transformations.

Corresponding Author:

Abdellatif Lasbahani,
RMI Laboratory,
FSTS Hassan 1st University,
26000. Settat, Morocco.
Email: abbdellatif.lasbahani@gmail.com

1. INTRODUCTION

In the resent years, we witnessed a fast technological evolution accompanied with a massive use of IT services that automated and facilitated many things in our daily lives. A tremendous amount of software applications and systems are built every day around the world to cover the evolving customers' needs. Consequently, existing applications should be updated and enhanced constantly. For this reason, Model-Driven Architecture (MDA) has been developed to provide an innovative development process based on models.MDA is an initiative proposed by Object Management Group (OMG) to pick up the standard UML, which is deployed for modeling Object-Oriented Systems' design. MDA tries to cover up the full software development life cycle starting from the inception phase until the maintenance or transition phase. In addition, MDA is a modeling language based principally on the sorting out of preoccupations in the form of abstraction levels where everyone represents a different system's view.

The separation of the system models can facilitate the improvement of the non-functional aspects with supplementary security information to make it more complete. Therefore, MDA approach promotes the massive exploit of model through the whole software development process pleading from a Computation Independent Model (CIM) dedicated for requirements specifications to code model or components transient by the Platform Independent Model (PIM) and Platform Specific Model (PSM). Code generation is performed by applying different kinds of translations or transformations: CIM to PIM, PIM to PSM, and PSM to source code.

This paper provides a new model-driven methodology, based on MDA, allowing the generation of secure software applications with functional and non-functional aspects after modeling them in the analysis and design phase. The implementation phase will be performed using Sequence Diagrams of the System'sInternal Behavior (SDSIB), which is used for modeling the system's internal behavior for each system's use case. In this approach, SDSIB will be deployed as PIM generated from CIM.

This approach improves the productivity of designers and developers, and the quality of service by allowing them to decrease the design's mistakes and deal with consecutive improvements, which are produced during the implementation phase. The application's internal behaviors are modeled through the sequence diagram of system's internal behavior that extends the system's functionalities. The internal behavior is developed together with its meta-model to illustrate the full transactions between objects during the execution of the use case. Each use case will be extended with the security policies that are already defined by the security experts at the specification phase.

The presented paper aims to carry on our earlier works by providing a Template-Oriented Code Generator that allows the generation of secure software applications through the new approach that extends the Java meta-model and sequence diagram meta-model to introduce the security concepts into the modeling and implementation phase. In this paper, we have chosen SDSIB as PIM because we found that this category of behavioral diagrams is better than behavioral diagrams due to its composition and offered services.

By applying a set of model-to-model transformations, we can generate automatically secure object-oriented software applications including the system's security infrastructure, methods and their bodies, persistence beans, configurations files, synchronization rules, business logic controllers, Data Access Objects, and security controllers. To do that, our generator uses, as input, an intermediate model, this is obtained directly from SDSIB after applying a set of model transitions. Such model aims to get better readability, scalability, reliability, and reusability of systems before and after the implementation phase. Additionally, such intermediate model is used to make bigger the target platform with non-functional improvements or Cross-cutting concerns, such as security policies properties, synchronization rules, persistence, or QOS constraints. In this work, we have tried to come together between functional aspect and non-functional aspect to consider the non-functional properties during the analysis, design, and the implementation phase.

For each use case, we affect the security profile to enhance the system's use cases with the indispensable permissions, such as confidentiality rules, traceability constraints, and non-repudiation rules, then each use case will be converted to SDSIB to determine different objects and sub-operations participants in the execution of the use case by following the new meaning of the Larman's Operation Contracts (LOC).

To generate secure object-oriented software applications, we have focused on scalability, maintainability, and safety of the system. We used Model View Controller (MVC) design patterns altogether with Grasp Patterns basing on SDSIB elaboration to feed the generator with the new-fangled structure of systems, which are three mains layers: views, controllers, and models. In this work, the controller layer will be subdivided into two controllers: Processing Controller (PC) and security controller (security validator) that will be obtained automatically through the proposed methodology.

SDSIB is gathered from Sequence Diagram of System's External Behavior (SDSEB) that represents only the actors' events and the system's responses. To protect each action or message initiated by the actor, we should apply the security properties verification before (precondition), during, and after (postcondition) the system's objects interactions, and before the data storage in the database by applying the novel tool so-called Extension Post- Pre Condition Larman (EPPL).

This paper is structured as follows. Section 2 summarizes related works. We give an Overview of the software development process and security engineering in Section 3. We show the proposed approach in Section 4. In Section 5 we talk about the sequence diagram of system's internal behavior. Results and discussionis given in section 6. We conclude our paper in Section 7.

2. RELATED WORKS

In this section, we will highlight the critical review of previous works that integrate monitoring strategies and security aspects into the software development process. specifically, the most important works that use MDA as a methodology to integrate non-functional aspects into the software development process.

In recent years, significant progress has been made in developing security techniques to address growing security concerns and performance. Among the most successful and best-known contributions in the world, we highlight work that is interested in the core work of Model Driven Security (MDS) to integrate security considerations into the software development process. First, David Basin and al. Basin D*et. al.* Introduce Model Driven Security (MDS) for the first time to design secure software [1]. To apply this methodology, the same authors proposed a solution to obtain the generation of access control policies from

abstract authorization rules that are written in the form of constraints. In addition, the proposed approach is more interesting and flexible at the same time.

InLodderstedt*et. al.*a domain-specific language (DSL) called SecureUML was deployed to take into account the security vulnerabilities and risks of the given technology in the development process[2]. This DSL is used to model access control policies through a new vocabulary to annotate UML models with the information needed for access control. However, this work remains very far from a generic methodology that supports all security properties and provides mechanisms for assessing sustainability.

With respect to security in service-oriented architectures, security objectives have been represented with graphical notations in business process models in order to transform security requirements into concrete security policies as outlined in [3]. Practically, the authors of this study have put forward a specific model called Security Policy Model (SPM), which is used to help implement security requirements such as authentication and access control. This work has not built specific tools to generate the final code corresponding to the security objectives and has no concrete formalism to describe the security needs throughout the software development process.

Satoh F *et. al.*Introduce a solution allowing annotating system's models with security rules about authentication[4]. In this work, authentication security policies generated by applying WS-Security Policy standard and Security Infrastructure Model (SIM) as a code template. this last one contains all security rules for the target platform; code template was deployed to map the business models with the security objectives. Additionally, an executable corresponding to security policies generated by applying this code template.

SatohF, Yamaguchi Ypropose an approach for generating the security configuration for IBM WAS[5]. This approach aims to affect the mapping between IBM WAS Deployment Descriptor (DD) and WS-Security Policy through a specific model deployed, which is used to ensure the necessary mapping between business models and security rules models; these models allowing transforming security policies to various security configurations such as IBM WA configuration. However, Juerjens and al. [6] propose a new approach so-called UMLsec as a solution to integrate the security aspects during the software development phases through the proposed guide in this work that help developers which are not expert in term of security to take into account the security risks and vulnerabilities in the system development stages. From a practical perspective, the proposed approach limited since it only resulted in the security specification modeling for a few properties without providing a methodology adopted and accompanied by a specific tool for code generation including the security rules according to chosen platform.

Next, a new framework so-called SECTET has been proposed by [7] for monitoring the security' engineering in parallel with the software engineering.This framework extended by a specific language called SECTET-LP for defining security policies in the software functionalities. Consequently, the abstract access control policies transformed to XACML code through XACML standard. From a comparative point of view, this approach confined to security exigencies declaration for a few properties without providing a concrete formalism to introduce the access control policies during systems development process. Also, no tool made in parallel for automating code generation for both aspects functional and non-functional.

Regarding component-based application security, [8] grant responsibility and permissions for the security expert to set up the high-level security policies. This work proposed a specific middleware called SecureMiddlware to extend CORBA component models with different necessary non-functional properties as the security or reconfiguration. In addition, security exigencies specified over the Policy Definition Language (PDL) and transformed immediately by an openPMF Policy Management Framework. However, this methodology does not support a specific tool for security configuration generation and does not guarantee the sustainability assessment. From a practical point of view, this approach stays immature to apply it to others technologies such as could computing, Multi-agent systems, and machine learning.

In spite of this, there no generic methodology for security integration and code generation from PSM, including genericity, re-utilization, and sustainability assessment. Regarding code generation, we cited approaches based on Petri-Nets [9] for automating code generation for functional concerns. However, this standard does not cover the modeling of the security aspect of system’s modeling and does not follow technological advances in term of security. Therefore, Petri-Nets cannot be a generic standard addressing security integration during all software development process, including analysis, design, and implementation.

To the best of knowledge, the most important methodologies given in this topic has not enriched with a specific code generator for automating code generation by deploying the existing tools to convert system’s functionalities to the source code. For example, Roubi S*et. al.* present a model-driven approach to generate GUI for Rich Application Internet by exploiting the using Eclipse Modeling Framework for Meta-Modeling, Query View Transformation for model transformations, and Acceleo for code generation [10]. However, this work does not address the code generation for both aspects functional and non-functional. Due to massive use of web services in different heterogeneous platforms, [11] propose a security interactive model of web service based on WebSphere and .NET to realize security interaction of heterogeneous

platforms. This model adopts a new approach based on predicate logic to integrate the security policies of heterogeneous platforms and uses the integrated policy to sign the SOAP message. Consequently, the proposed method is specific for web services and can not be applied to others technologies.

As far as cloud implementation is concerned, technological advances in security also target cloud computing or cloud architecture to secure and secure the cloud platform. To this end, much work has been done in this direction, but we are discussing only the most important contributions such as [12], which deal with security and safety in cloud implementation by providing guidance. users with cloud and mobile cloud expertise to help customers choose cloud implementation appropriately for security and safety constraints. However, this approach does not provide a method for describing the timing and integration phase of the security architecture, and does not provide a specific tool for generating cloud implementation from the cloud architecture.

In this current work, we completed our previous works given on this topicby monitoring data flows during the exchanging of messages between software's components firstly and increasing the number of security properties addressed. In this work, SDSIB was chosen as a PIM instead of communication diagram [13]-[17] with the aim to apply the new semantics of the security during the elaboration of the software operations in the goal to apply the new semantics of the security during the elaboration of the software operations.

We have implemented our methodology to address service qualities, including the generalization of automation in all aspects, reduced application development costs, decreased maintenance costs, and sustainability assessment. The new syntax was obtainded from LOC In order to improve SDSIB with the way of exchanging messages between software components, this semantics feeds UML with precise semantics to facilitate code generation according to the chosen platform.

This paper is underwritten in security engineering and model-driven engineering, which aims to cope with the increasing rise of the Internet of Things such as objects, components, agents or Web services. This MDA-based contribution and the UML profile that extends the intermediate model with security constraints; this type of model is used to allow platform scalability compared to other enhancements that can not be formatted with the system's modeling languages and to improve the software architecture before the implementation phase.

3. OVERVIEW OF THE SOFTWARE DEVELOPMENT PROCESS AND SECURITY ENGINEERING AUTOMATION

In this section, we present an overview of the Larman'soperation contractsand the syntax of the security constraints. We show the SDSIB meta-model and the various improvements that have been made over the SDSIB to take into account the security semantics of security properties during flow interactions between software components. We describe the most important transformations performed to improve our intermediate model of the chosen platform with security constraints such as authentication rules, authorization rules, integrity rules, data encryption rules.

3.1. Overview of Larman's Operation Contracts

A Larman's Operation Contracts (LOC)[18]describe the state of the system before and after receiving or calling on system components by system actors. From a semantic point of view, LOC describe how the components of the system cooperate with each other to respond to requests by determining the different objects involved in the execution of the data flow. This contract defines all collaborations to execute the requested use case. condition and post-condition. From a logical point of view, Graig Larman was interested in the internal behaviors of the system through the definition of the state of the system before and after the execution of the request in order to build the design of the system. In the pre-condition, LOC describes the initial system. state, while the post-condition describes the state of the system after completion of the query.

Specifically, Graig focused on the state of the system by describing many things such as created objects, destroyed objects, formed and broken associations, and changes in the state of attributes after the execution of the query. In the pre-condition, GraigLarman was addressed to the system state before reaching the requested component. From a safety point of view, Graig does not describe the state of the system in terms of safety and sustainability assessment. Semantically, security rules are not retrieved before and after the completion of the target resource. The old LOC does not correctly and securely determine the main objects or components involved in executing requests from system actors.

To do this, we proposed an extension of this contract to become versatile and cover security issues at the same time. Therefore, the semantics of security policies described by security experts will be considered in the new definition of LOC to semantically cover them in the state of the system before and after the execution of the process. Like that, we integrated the security properties during the software

development process. During the extension of LOC; we benefited from the benefits of the MDA approach to reduce design errors and vulnerabilities.

Thanks to LOC's new semantics, we can safely draw and generate all the interactions between the system components involved in constructing the response. Practically, SDSIB has been dragged on the basis of the new LOC definition, and the Object Constraint Language (OCL) used to describe the constraints of security policies. SDSIB construction will be based on GRASP patterns to facilitate the assignment of responsibilities to participants involved in the execution of use cases.

3.2. Overview of Pre-Post Condition Extension

In this section, we have extended the pre-post condition of LARMAN to introduce and validate the properties of security policies such as authorization, authentication, data integrity, data encryption and non-repudiation when developing operating contracts. The main purpose of this extension is to inject the security constraints, during the change of state of the system, when an action is triggered by an actor.

This new semantics of LOC allows designers to draw a secure SDSIB in which we designate all interactions between interacting objects to respond to incoming requests. These improvements reduce design errors and improve design quality. In other words, we guarantee that source and target objects interact securely and respect the security constraints defined in the UML profile.

In this extension, we have incorporated the GRASP patterns after their extension to distribute the assignment of tasks on the system objects. It is used to initialize the prerequisites of the initial state of the system and to describe the state of the system after execution of the use case postconditions. We used OCL to enhance system functionality with security policies; precondition and postcondition. Conversely, the new semantics of LOC allows developers to securely conclude the various operations including source and target objects participating in each interaction. Via a specific code generator, we automatically generate system operations code along with their security policies.

3.3. Overview of Extended Precondition

In the extended precondition, we introduced the security policies constraints verification to check system's state in term of security such as authorization, authentication, and data integrity before continuing to achieve system' use cases. To produce secure software applications, it is recommended to describe the semantics of the security policies during the software' architecture design to facilitate UML profile application on the intermediate model. Our methodology applied to the existing software development methodologies like UP-UX for extending them in the goal to consider both the functional and non-functional aspects at the same time during the system design. After being extending the SDSIB with the UML profile, we apply our proposed tool or code generator to generate the code from the system's models. For this reason, the code generator should be able to interpret the new semantic of LOC and generate the corresponding code including system methods, methods bodies, class attributes, security infrastructure, security controller, and DAO. As a result, the non-functional aspects configurations will be decreased automatically according to the chosen platform by providing the functioning rules definition concerning the user permissions and data flow protection.

By applying the new definition of LOC, designers and developers can benefit from numerous advantages which are:

1. Reducing design errors.
2. Reducing the system security.
3. Improving the system's qualities.
4. Keeping a reverse-engineering during system design.
5. Improve the designer's knowledge in terms of security techniques.
6. Verifying the security policies constraints related to system resources before using them.
7. Improving the modeling quality.

3.4. Security Constraints Syntax

In this section, we provide monitoring mechanisms to enhance the static and dynamic behavior of systems with the security constraints of authentication verification, authorization verification, data integrity verification, and encryption verification. critical data by applying the UML profile. improved with the necessary security constraints, which are described by OCL. Then, these constraints and any other enhancements will be applied on the SDSIB to assign security requirements and responsibilities to the objects involved in the execution of a system action. To take into account the code generation of the infrastructure of the system. security, with the security templates applied to the functional aspects when generating the code according to the chosen platform; the generated code will be a language understandable by the compilers.

By applying a set of model transformation, the proposed generator generates a built-in security controller (SC) to check the semantics of the security policies represented by the stereotypes during the software development process; the security policy infrastructure generated from the SDSIB in the form of XMI / XML.

3.5. Security Policies Constraints Syntax for Permission Monitoring

To interact with software components, authorization monitoring is recommended to check the status of the logged-on user's identity based on the security policies involved in modeling the requested components. In our methodology, authentication and authorization security policies are strengthened to ensure the durability of unsupervised requests through security policy constraints.

To do this, we proposed the syntax of the security constraint to reinforce user authentication and authorization via the OCL to restrict unauthorized access as shown below.

```
Context Object_Name
Inv: list_users->includes(connected_user)
```

In addition, each class stereotyped with a secure stereotype, automatically a preliminary authentication is triggered before interacting with the requested resources.

In this approach, we applied security checking on the content of system objects themselves such as methods, attributes, associations, resources, data flows between objects, and implemented interfaces to decrease system security. To do this, we have profiled the system objects, operations and attributes with the already improved UML profile with the security policy constraints proposed by the security experts. These constraints are assigned to the UML profile according to the following syntax, as shown below.

```
Context Object_name:: Method_Name(Parametrs):: Return_Type
pre: Self.Operation_Name.Permissions->NotEmpty
and self.attributes->NotEmpty
and Connected_User.role.permissions>includes
(self.Operation_Name.permissions)
and View_OR_URL.permisions->
includes(connected_User.role.View_Name_OR_URL.permissions)
and self.associationEnd.permissions->includes(connected_User.permissions->select(permissionsassociation.
permissions))
```

In our UML profile, we have developed a specific meta-class (stereotype) to apply security policies on the attributes of system objects. For this, we have improved this stereotype with the security constraints of the policy as shown in the following syntax. As a result, this solution helps reduce security vulnerabilities and improves the quality of the generated systems.

```
Context Object_Name
Pre: self.attributes>forAll(attribute | attribute->NotEmpty)And
Connected_User.role.permissions>includes(self.attributes.permission)
And primitivesTypes->exists(self.attributes.type)
And Coonected_User-> size () = self.attrinutes. multiplicity
```

In this methodological support, we have proposed another meta-class to reinforce data encryption during interactions between software components. This meta-class enriches with the security policy constraints on data encryption as shown in the following syntaxes; the first applied to the attributes of the object and the second to the entities. As a result, the code generator enriched with a specific template dedicated to the adoption of profiled models with existing data encryption algorithms.

```
Context Object_Name
Inv: self.attributes.value->NotEmpty
post : self.attribute.value=convert(self.attributr@pre)
```

```
Context object_Name
Inv: self.value->NotEmpty
Post: self.value@pre=self.value@post
```

In order to validate these constraints, we applied the security profile after its extension to the security constraints on the SDSEB as shown in Figure 1. We implemented the proposed generator to generate the source code corresponding to the two functional and non-functional aspects by their templates described through the helps.

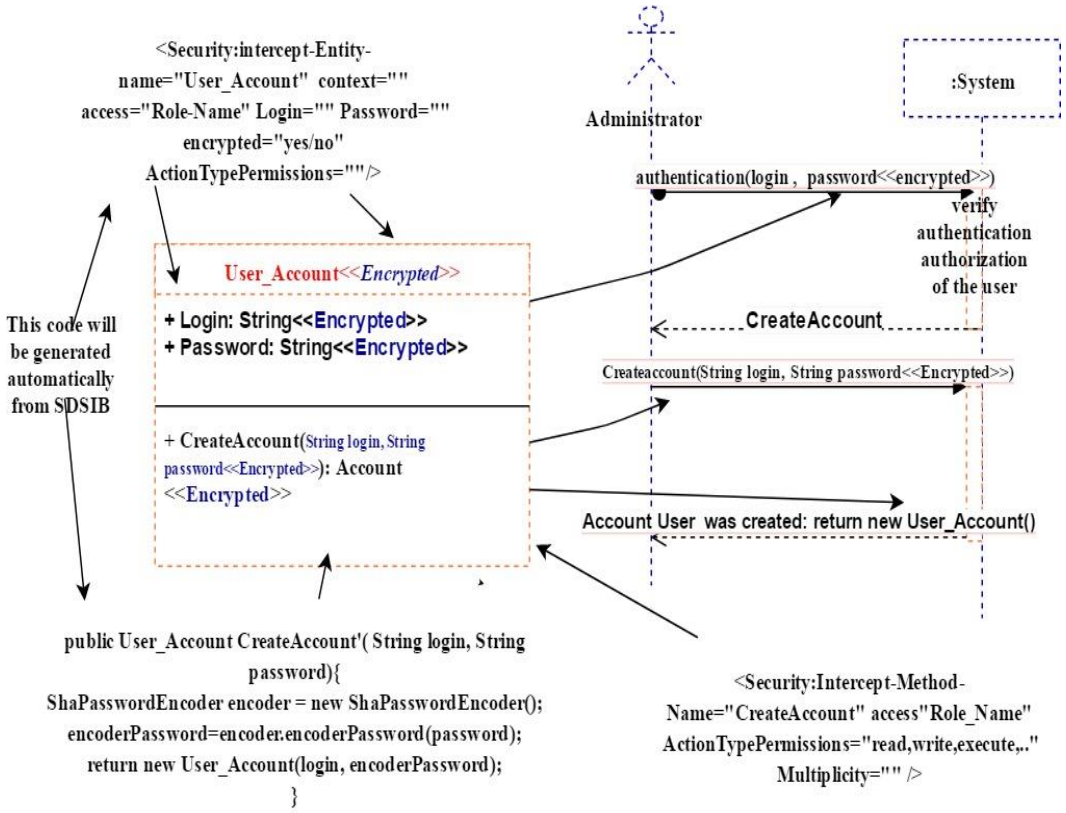

Figure 1. SDSEB profiled with Encrypted stereotype

3.6. Data Integrity Constraints

Semantically, we have strengthened our methodology to approve the state of the object in terms of data integrity when developing prerequisites and post-conditions. We have developed specific stereotypes to promote software components with data integrity issues. These stereotypes apply to the intermediate model and its content, which is derived from PSM to increase software requirements with data integrity concerns. To facilitate the construction of data integrity constraints, we have suggested a syntax for checking the integrity of object data, as shown in the following syntax. The code generator enriched with a specific model dedicated to data integrity for the conversion of profiled models into source code.

```
Context Object_Name
Pre: self->NotEmpty
Post: self.attributes.values->
forAll( value | attribute.value =attribute.value@pre)
```

Therefore, we can affect security policy monitoring by applying the new LOC definition and more specifically during SDSIB development. We obtained SDSIB from the SDSEB by applying the new LOC to determine all the components involved in performing software actions and to support the security architecture at the same time by applying the UML profile.

We have expanded the GRASP models to assign responsibilities and security policies to the software components. The new semantics of GRASP allow us to precisely define the system resources needed to complete the business processes of the system by recognizing the constraints of low coupling and strong cohesion.

3.7. Grasp Patterns Concept

Craig Larman has created Grasp patterns to solve modeling errors. The purpose of grasp patterns is to provide a mechanism for assigning responsibilities to solicited system objects to execute system processes. In fact, there are nine input models that are: low coupling, high cohesion, designer, expert, controller, pure manufacturing, polymorphism, information expert and protected variations. In our methodology, we have extended them to take account of the samanticsand constraints of security policies when assigning responsibilities by improving the low coupling and high cohesion. In addition, we have added another pattern to the existing patterns, which is the security controller deployed to monitor infrastructure security according to the chosen platform and put in place considerations of low coupling and high cohesion; this controller is positioned at the front to intercept the status of queries coming from outside. In Table 1, we show the new semantics of some models.

Table 1. Grasp patterns and their new semantic

Grasp pattern	Description	New semantic
Low coupling	Minimize the links between system's components to facilitate their scalability, readability, reliability, and reusability.	Reinforce the system's components to verify the security policies proposed through the UML profile before establishing a new link between system's objects.
High cohesion	Reinforce the system's objects to realize the actions belong to its responsibilities.	Improve the system's objects responsibilities with the security policies conditions illustrated by the UML profile; these constraints checked out through the design pattern: security controller.
Security Controller	Dedicated to separate between the system's layers and facilitate the system's scalability.	We have proposed a new design pattern sacrificed for monitoring the non-functional properties to improve the low coupling and high cohesion

By applying the new semantic of grasp patterns, we can introduce and verify the security policies constraints during the assignment of responsibilities to the objects involved in achieving the system's actions. Designers enrich system's functionalities with the security policies exigencies by applying the new definition of grasp and the security profile.

3.8. Overviewof Extended Postcondition

In this work, we extended the LOC postcondition to consider the validation of security policies described in the prerequisite. Systematically, the postcondition defines the basic state of the system after execution of the system use cases. This extension allows domain experts to improve the functionality of the system with the semantics of security policies and more specifically the creation of objects, the destruction of objects, the formatting of associations, the destruction of associations and the change of state after the execution of the request. Therefore, domain's experts use our UML profile to improve the SDSIB with the security policies constraints about authentification, authorization, data integrity, availability, and data encryption.

4. PROPOSED APPROACH

In this contribution, we present a new approach for modeling the software architecture and integrating the most important security considerations into system design. Our intention is to overwhelm security vulnerabilities and increase the scalability of security. We used our methodology for the UP-XP approach to overcome non-functional features throughout the software development process. We encouraged this approach with a particular code generator to produce the final code from profiled models. As a result, software developers accurately produce the internal behavior of the system and act on the low coupling and high cohesion to provide readable systems as shown in Figure 2.

To do this, we used MDA as a modeling language that supports the extended use of models through model transformation. These changes allow architects and developers to manage the various changes produced in the customer's needs and to automate the maintenance and evolution of existing systems via the mapping rules between the two functional and non-functional aspects.

First, we generated the SDSIB from SDSEB after applying the new LOC in order to correctly determine all the interactions and system components involved in the execution of the system processes. In this model, we assign the responsibilities and security semantics to system's objects.

This model is considered a Platform-independent Model (PIM) generated from the Computing Independent Model (CIM) model by applying a set of model-to-model transformations; CIM focuses on the system environment. Then we produce the PSM by adding the most important information related to the chosen platform (platform models) to PIM. The PSM already improved with the security policies semantics. Therefore, we must improve it with the constraints of security policies depending on the chosen platform. To do this, we have allowed PSM extensibility by deducting an intermediate model from PSM; the intermediate model is ready to receive other improvements related to a non-functional aspect.

In addition, we have stereotyped this model with our UML profile (security profile) to grow it with the constraints of security policies. The UML profile for security is an extension of UML dedicated to improving the software's functionalities with security features related to authentication, authorization, data integrity, and data encryption; The UML profile is based on stereotypes and values marked to adopt UML for a particular performance.

Practically, we have proposed a specific code generator for the UP-XP approach to generate the source code corresponding to both functional and non-functional aspects. This code generator takes as input an already annotated intermediate model with security policy performances and converts it into source code. It uses their templates to generate the final code by weaving the security policies on the software's functionalities through these templates; a template described by helpers.In Figure 3, a diagram describing the owerview of the approach. We also illustrate the most important stations for the production of secure and scalable applications.

In Figure 4, we present a current example describing how to automatically generate the complete code for system's entites, attributes, operations, persistent entities, and security configuration. The source code deduced from the design model obtained from SDSIB by applying a set of model transformations. In this conversion, we enriched the domain class diagram with the necessary operations derived from SDSIB already enhanced with the semantics of security policy constraints. Then, we applied the security profile defined by UML profile to increase the security of secure semantics. After having been stereotyped the intermediate model, the code generator intervenes to weave their templates into the system's functionalities. As you can see, we annotated Email, Login, and Password respectively with EmailValidation, unique and encrypted stereotypes. As a result, these stereotypes extend these attributes with the security policies constraints.

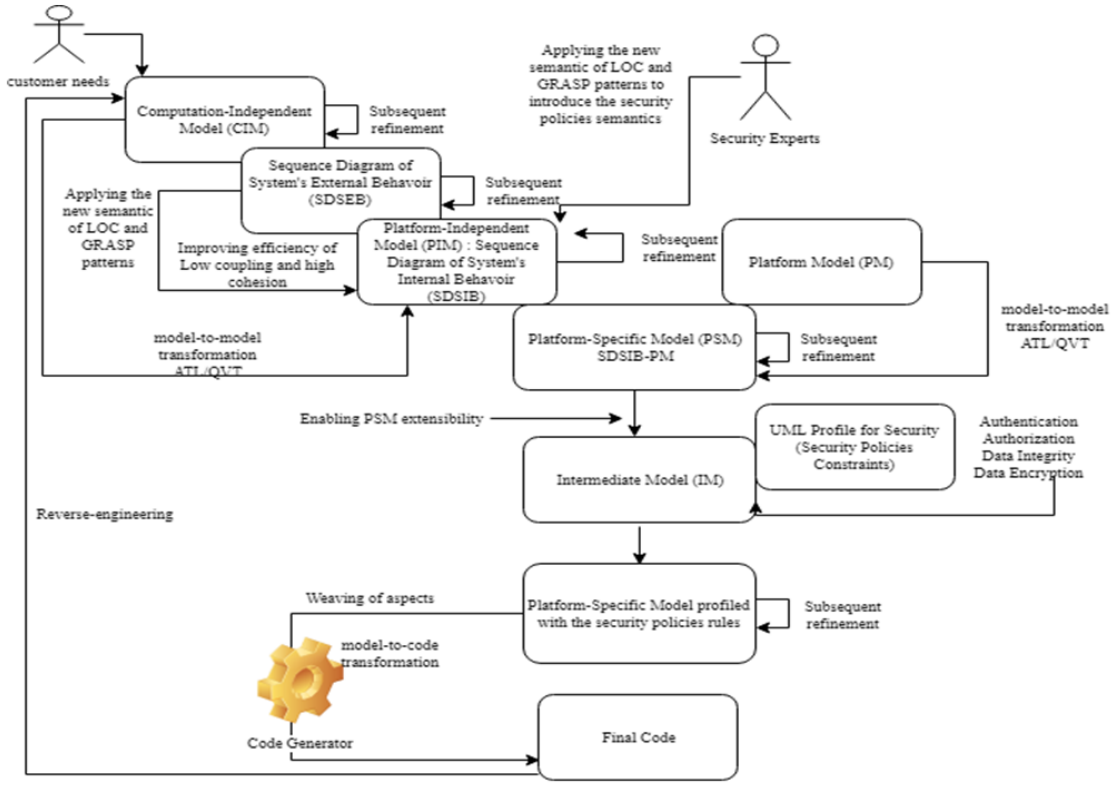

Figure 2. Diagram describing the owerview of the approach

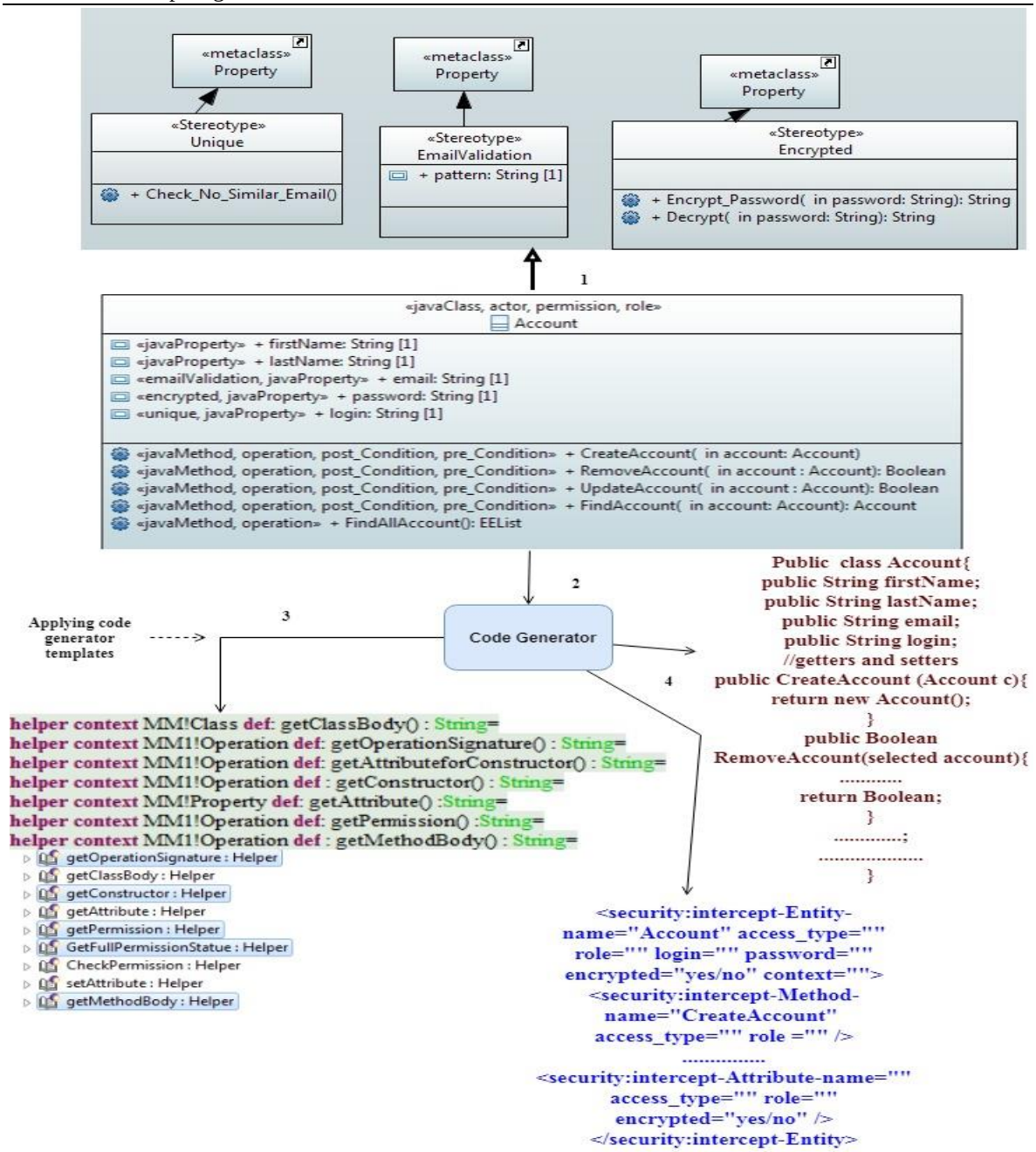

Figure 3. Running example of code generation from profiled model

5. SEQUENCE DIAGRAM OF SYSTEM'S INTERNAL BEHAVOIR

Sequence Diagram of System's Internal Behavior (SDSIB) is a sequence diagram got from the Sequence Diagram of System's External Behavior (SDSEB) after its enlargement with the new semantic of LOC and grasp patterns. Our overall strategic objective is to indicate responsibilities to system entities through the new extension of design pattern high cohesion, security controller, expert, creator, and low coupling. We have updated these responsibilities to take into account the security context suggested by the chosen platform described via the UML profile for security. This technique will promote the integration of security policy constraints at the intermediate model level. In addition, we have reinforced this commitment to describe the state of the system before and after executing system actions in a secure manner. In this SDSIB, we cite precisely all the components of the system, and the interactions negotiated to perform a requested action; SDSIB focused on the temporal aspect of the message transaction between system objects.

To provide an approach that considers non-functional aspects during the software development process to produce scalable and secure software, we provide potential enhancements to the SDSEB to inject the semantics and structure of the non-functional aspects during their elaboration. The new SDSIB

representation allows domain's experts to draw secure communications between system's objects according to security responsibilities granted to each component participated in the interaction. The new semantic of LOC and design patterns allow us to take into account authentication rules, authorization rules, integrity rules, availability rules, and data encryption rules at the assignment of responsibilities to contributed objects. Then, we improve them with the requested stereotypes to consider the performance of the UML profile; this is done after the integration of the platform models.

5.1. Proposed Metamodel for SDSIB

We chose the SDSIB as PIM, so we defined the PIM meta-model by adopting security performances and sustainability during the whole software lifecycle. As shown in Figure 4, we have reached the SDSEB to point out the operations and resources of the system planned to achieve the final goal which is the security policies integration, and automatic generation of code from secure models. We applied the new semantics of LOC and grasp patterns on the SDSEB to extend it to SDSIB and integrate the low coupling and, high cohesion performances.

For use cases of the system, we draw a SDSIB to define the state of the system before and after the treatment by following the proposed meta-model that describes the structure of the interactions. We thus enrich SDSIB with the necessary meta-classes of security meta-models such as ModelElement, Permission, Role, ModelElement, ActionType, ImplementationEnvironment, ActionContext, Precondition, PostCondition, IntegrityConstraints, AthentificationConstraint, AthourizationConstraint, DataEncryptionConstraint, and AvailabilityConstraint.

We cite components, business controllers, security controller, DAO, and view layer for each use case. By applying a set of model transformations, we incorporate the security policies meta-classes designated in the security meta-model to SDSIB. Accordingly, we strengthened introducing of security policies semantics during SDSIB discussion in order to improve its extension at the intermediate model level via the UML profile.

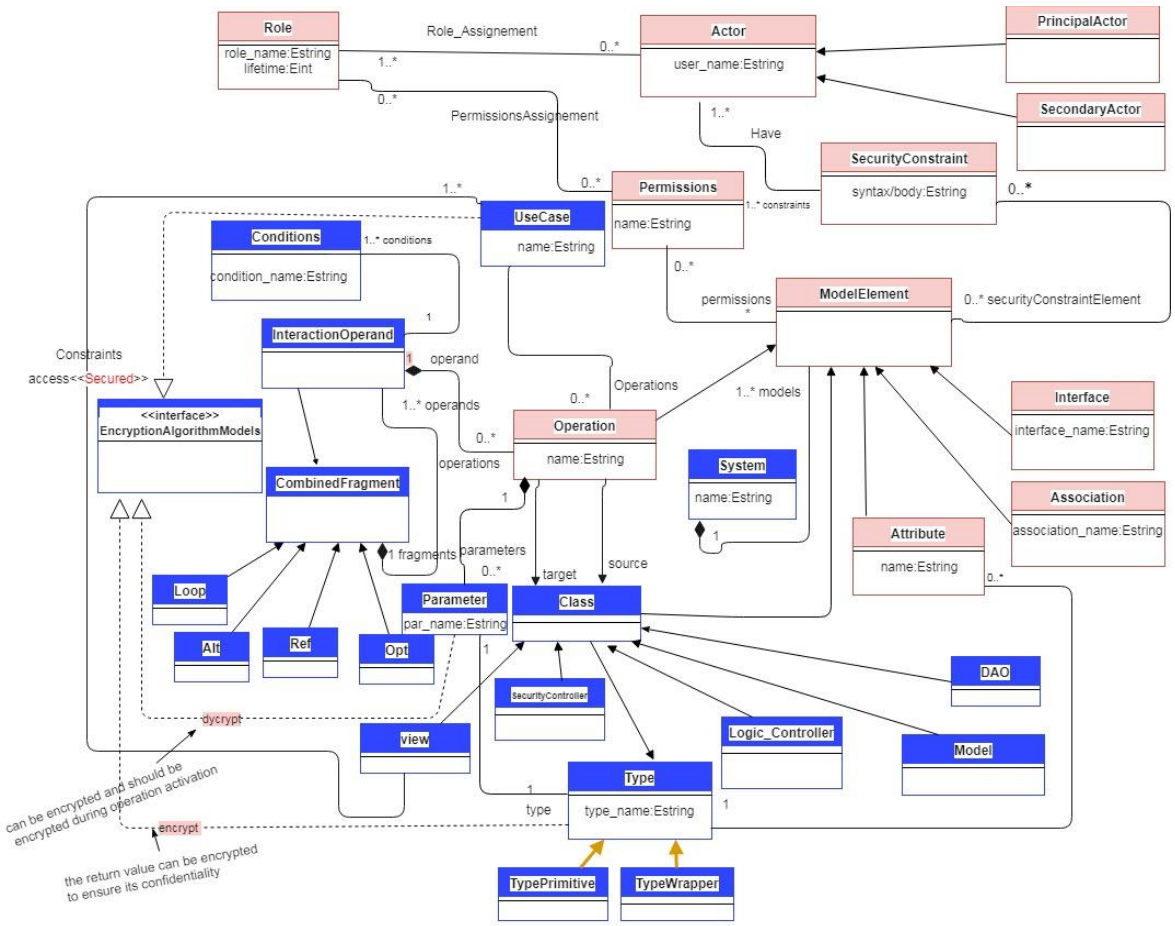

Figure 4. Proposed metamodel for Sequence diagram of system's internal behavior

5.2. Transformation Process

5.2.1 Model to Model

Once the modeling phase built, we set up the transformation rules in the goal to connect each element of the source model with its equivalent in the target model according to the conditions and rules for converting all necessary changes. It is deployed to make a subsequent refinement called exogenous transformation. We can use it to make the correspondence between all elements of the security policies model of the input model and the element of the SDSIB output model. It does the correspondence among all components of the SDSIB input model and the components of the domain class diagram output model to improve it with the system's internal behavior. For each interaction, we define the semantics of security policies, precondition, postcondition, and solicited components. Then, we connect these interactions to intermediate model to extend it with necessary improvements provided via the UML profile for security; we specified Atlas Language Transformation (ATL) [19] as tool.

5.2.2 Model to Text

The first conversion produces the XMI/XML file for the design model respecting the new semantic of LOC and design patterns. The second step consists in generating a final code of the software from the profiled model. To do this, there are numerous mechanisms for code generation; deploying scripting languages, XML pressing, WebML, WebDSL, Stratego/XT or Acceleo. However, these tools have not focused on generating source code for no-functional aspects and do not use an intermediate model for enabling the PSM extensibility before code generation. Additionally, we proposed our own code generator based on the model transformation language.

5.2.3 Code Generation

For automatic code generation for both aspects functional and no-functional, we proposed our own code generator that is implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard. In this tool, we considered the following criteria:

1. Do not use concrete syntax.
2. Uses intermediate model.
3. Supports the entire development process.
4. Supports Object-Oriented systems.
5. Generates system'soperations and their bodies.
6. Generates security infrastructure configuration.
7. Generates security controllers.
8. Generates controllers.

The operating principle is to expand generated templates to produce the last code from output model profiled with the UML profile (intermediate model). We preferred Java as a platform to approve our approach. So we established specific templates for the security policies in term of authentication, authorization, data integrity, data encryption, and availability through the helpers written in ATL. Then, we improved the business logic code with other layers of security policies given respecting the code generator templates; the code generator templates automatically transform the profiled models obtained in the previous transformations into source code according to the chosen platform.

6. RESULTS AND DISCUSSION

As result, we have suggested a new methodology to lead the whole development process based on the separation of concerns. We have strengthened the MDA theory to incorporate the security application and software durability throughout the software development cycle by determining the correct time of improvement. For this, we have extended the MDA structure to combine absorptions and safety performanceas an independent abstraction. We offered a new semantic for LOC and design patterns as a methodological support for producing software's behavior.In this semantics, we have improved the effectiveness of low coupling and high cohesion of design patterns to promote software scalability.We established a set of syntaxes for security policies written in OCL to validate the relevance of systems models.we applied this LOC on the SDSEB to obtain an SDSIB improved with security rules and scalability of the software, security policies granted to the responsibilities of the object.

From a statistical point of view, our methodology increases the degree of automation, the level of genericity and reuse, and the number of lines of code generated in comparison with other approaches given in these topics. We addressed five properties: confidentiality, authorization, availability, data integrity, and data encryption while there are many important approaches addressing just access control. In addition, we proposed the UML profile for security as a specific tool to improve the intermediate model with security policies constraints according to security needs, and automate the implementation phase. This approach has been enhanced with a formalism facilitating the integration of security policies.

From a practical point of view, we produced our own specific code generator based on the model transformation language to generate the corresponding code form profiled models. The value added by this proposal is that we use an intermediate model to enable PSM extensibility before generating the source code. Therefore, this advantage does not appear in previous code generators as WebML, WebDSL, Stratego/XT or Acceleo. Existing code generators and methodologies do not generate security configurations and do not support the entire development process.

7. CONCLUSION

In this paper, we proposed a new model-driven approach for generating secure software. This methodology based on the OMG standard, the new meta-model for SDSIB, UML profile, and new semantic of LOC and design patterns. To establish this approach, we first study and extended the SDSEB to SDSIB to implement the semantics of security policies by applying the new semantic of LOC which is an implementation of LOC and OCL; the generated model represents our input model (PIM) which also describes security policies and interactions. We have defined a meta-model for the SDSIB taking into account the constraints of security policies when assigning responsibilities. Secondly, we inject the platform models to obtainded PIM to set up a specific PSM; we chose as an implementation the java platform. third, for reasons of extensibility we retrieved an intermediate model from (PSM) to allow its extensibility. In this work, we profiled our intermediate model with the UML profile for security to improve the semantics of security policies with the security rules communicated through a set of stereotypes. After that, the code generator was developed for both model-model (refinement) and intermediate model to text (source code). As result, we were able to generate secure applications that respect the new LOC semantic and design patterns. We were able to combine between both aspects fuctional and no-functional throughout the software development process.

As future studies, we intend to generalize this methodology on.NET, CORBA, and PHP platforms and address other properties of non-functional aspects such as persistence, synchronization, and the component-oriented systems. Also, we aim to enhance our generator to cover the view layer by generating the Graphical User Interface (GUI) for each use case based on the chosen platform.

REFERENCES

[1] Basin D, Doser J, Lodderstedt T. "Model Driven Security: From UML Models to Access Control Infrastructures". *ACM Transactions on Software Engineering and Methodology*.2006; 15(1): 39-91.
[2] Lodderstedt T, Basin D.A, Doser J. "Secureu ml : A uml-based modeling language for model-driven security". *Proceedings of the 5th International Conference on The Unified Modeling Language*. London, UK. 2002;2: 426-441.
[3] Wolter C, Meznel M, Meinel C. *Modeling security goals in business processes*.Köllen. 2008; 127:201-216.
[4] Satoh F, Nakamura Y, Ono K. "Adding Authentication to Model Driven Security". *IEEE International Conference on Web Services*. Chicago, IL, USA. 2006; 127: 585-594.
[5] SatohF, Yamaguchi Y. "Generic security policy transformation framework for ws-security". *IEEE Computer Society on ICWS*. Salt Lake City, UT, USA. 2007;92: 513-520.
[6] Juerjens J. UML sec*:* "Extending UML for secure systems development". *Proceedings of the 5th International Conference on The Unified Modeling Language*. London, UK.2002; 2460: 412-425.
[7] Hafner M, Breu M, Breu R, Nowak A. "Modelling inter-organizational workflow security in a peer-to-peer environment". *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*. Washington, DC, USA. 2005;3292: 533-540.
[8] Reznik J, Ritter T, Schreiner R, Lang U. "Model driven development of security aspects". *Electronic Notes in Theoretical Computer Science*. 2007;163(2): 65-79.
[9] Girault C, Valk R. *Petri-Nets for Systems Engineering*.Berlin.2003.
[10] Roubi S , Erramdani M, Mbarki S.A, "Model Driven Approach based on Interaction Flow Modeling Language to Generate Rich Internet Applications". *International Journal of Electrical and Computer Engineering (IJECE)*. 6(6): 3073-3079, 2016.
[11] Tao X, Xin H, JiwenX, Shujuan S. "Security Interaction of Web Services in Heterogeneous Platforms". *International Journal of Electrical and Computer Engineering (IJECE)*, 2014; 12(4):2868-2874.
[12] Mahmoudi C. "Cloud and mobile cloud architecture: security and safety". *Handbook of System Safety and Security*. 2017; 10:199-223.
[13] Lasbahani A, Chhiba M, Tabyaoui A, Mjihil O. "Model Driven Architecture Approach for Application Security Integration". *Journal of Theoretical and Applied Information*, 2017; 95(8): 1655-1668.
[14] Lasbahani A, Chhiba M, Mjihil O. "Deals with integrating of security specifications during software design phase using MDA approach". *Proceedings of the Second International Conference on Internet of things and Cloud Computing*. Cambridge, UK.2017; 196: 1-7.

[15] Lasbahani A, Chhiba M, Tabyaoui A, Mjihil O. "MDA Approach for Application Security Integration with Automatic Code Generation from Communication Diagram". *International Conference on Information Technology and Communication Systems*. Springer. Khouribga, Morocco. 2018; 640: 297-310.
[16] Lasbahani A, Chhiba M, Tabyaoui A, Mjihil O. "A New Extension of Larman's Operation Contracts for Security Properties Injection and Verification during the System's Internal Behavior Elaboration". *Proceedings of the 2nd International Conference on Computing and Wireless Communication Systems*. Larache, Morocco. 2017; 33: 1-6.
[17] Lasbahani A, Chhiba M, Tabyaoui A. "A Model Transformation Methodology for Security Integration and Code Generation from Sequence Diagram of System's Internal Behavior". *International Review on Modelling and Simulations (IREMOS)*. 2018; 11(2): 102-116.
[18] Larman C.Applying UML and Patterns. 3rd Edition. Prentice Hall. 2002; ISBN 0-13-148906-2.
[19] ATL - a model transformation technology. http://www.eclipse.org/atl/. 2012.

BIOGRAPHIES OF AUTHORS

Abdellatif Lasbahani (corresponding author) was born in El Attaouia, Morocco, in 14/04/1989.He received the Master in engineering design and applications development from faculty of sciences and techniques, Settat, in 2014. He is currently working as PhD student at university of Hassan 1st Morocco. His area of research includes the security integration and the code generation from UML models for automating the software development engineering and generating the secure software applications from software design models.

Mostafa Chhiba was born in Kenitra, Morocco, in 13/12/1963.He has obtained his PhD in Molecular Sciences from University Sciences and Technologies Lille1, in Jun 1995. He is currently working as Professor at faculty of sciences and techniques, Hassan 1st University. His area of research includes simulation and molecular modeling.

Abdelmoumen Tabyaoui was born in Errachidia, Morocco, in 15/04/1965. He obtained his Phd in Molecular Physics from Burgundy university, Dijon France 1992, and a Doctorate thesis in Physical Sciences from the Mohamed Vth University of Rabat, Morocco, in 2005. DR Tabyaoui, professor at the Faculty of Sciences and Techniques, Hassan 1st University, is currently working as a Deputy Director of the National School of Applied Sciences of Khouribga, Hassan 1st University, and Settat, Morocco. His research area includes atomic and molecular physics.