Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 443–452 ISSN 2300-5963 ACSIS, Vol. 39
DOI: 10.15439/2024F1796

MBSPI—A Model-Based Security Pattern Integration Approach for software architectures

Anas Motii
0009-0005-4936-0028
Mohammad VI Polytechnic University
College of Computing
Benguerir, Morocco
Email: anas.motii@um6p.ma

Mahmoud El Hamlaoui
0000-0003-3315-7373
Mohammed V University in Rabat
Rabat, Morroco
Email: mahmoud.elhamlaoui@ensias.um5.ac.ma

***Abstract*—Incorporating security patterns into software architecture is essential for robust system design. Model Driven Engineering (MDE) offers a structured approach to software development, emphasizing modeling and automation. This paper explores the integration of security patterns into software architecture using MDE techniques. We highlight the benefits of this approach, including improved level of security and enhanced maintainability. Challenges such as modeling complexity and tool support are also discussed. Through a SCADA (Supervisory Control and Data Acquisition) system case study, we demonstrate the effectiveness of integrating security patterns into software architecture using MDE.**

***Index Terms*—Security patterns, Pattern Integration, Software architecture, MDE, OCL**

I. Introduction

IN THE realm of system and software architecture, while there exists a fundamental understanding of security engineering principles, there is often a notable gap in best practices necessary for implementing security measures derived from risk assessments. This gap has propelled the exploration of security patterns as a prominent field of study in recent years. Security patterns serve to encapsulate and disseminate expert knowledge by offering generic, reusable solutions to frequently encountered security challenges, particularly those related to architecture. This methodological approach seeks to equip architects with the tools required to effectively address and mitigate common security vulnerabilities. A complete catalog of security patterns has been introduced by Fernandez [1]. Unfortunately, there are two major issues. First, traditional security patterns are usually described as informal guidelines to solve a certain problem using templates such as POSA (Pattern-Oriented Software Architecture) and GoF (Gangs of Four). Despite the benefits of security patterns in promoting reuse, their practical application faces significant challenges. The disconnection between the threat models identified through risk assessment and the protective measures outlined in security patterns creates a substantial barrier. Additionally, the manual integration of these patterns into architectural designs introduces further complexity, often leading to incorrect implementations. This misalignment not only hampers the effective utilization of security patterns but also leaves critical security issues unresolved, underscoring the need for improved methodologies in pattern integration and application. One notable finding in [2] is the lack of emphasis on pattern integration, which served as a key motivator for our research endeavors.

Drawing on the advantages of Model-Driven Engineering (MDE) such as improved quality and productivity, we use dedicated modeling languages and Model-To-Model techniques tailored to conduct a comprehensive pattern integration process. In this paper, we introduce a Model-Based Security Pattern Integration (MBSPI) approach for software architecture and its tool support. We use the Object Constraint Language (OCL) for the formalization of security properties.

The remainder of the paper is organized as follows. Section II identifies related work to pattern integration. Section III presents the main steps of the MBSPI approach. The MDE framework is described in section IV, more specifically, Model-to-Model transformations, and OCL constraints. Section V specifies requirements for tool support. In section VI, MBSPI is assessed over a SCADA (Supervisory Control and Data Acquisition) system case study. Finally, section VII, concludes and sums up the contributions and future work.

II. Related work

Over the years there has been a noticeable divorce between pattern experts and pattern users [3]. On one hand, pattern experts create and document patterns and on the other hand, pattern users are rarely aware of relevant patterns. In addition, the latter do not have a good understanding of how to leverage and apply a pattern. Works relevant to the integration of patterns and aspects are discussed since this issue (i.e., integration) has been tackled in both research areas.

In [4], the authors explained how pattern integration can be achieved by using a library of precisely described and formally verified solutions. In [5], the authors present an approach for creating a security-enhanced system model using the SecFutur Engineering Process and the SecFutur Process Tool (SPT). In [6], the authors introduced a method and tool support for developing secure and private IT systems using COmputer Supported Security Patterns (COSSP). The integration process targets object-oriented applications.

Aspect-Oriented Modeling is similar to pattern modeling with regards to encapsulating concerns such as security for use. However, the difference is that aspects are part of software fulfilling a function dealing with the design stage, whereas patterns can deal with different development stages. In [7], Nguyen et al. presented a pattern-driven secure system development process combined with an aspect-oriented security design methodology. To use this approach, the designer is required to manually construct security solutions of the considered system and the definition of mappings of these solutions into the model under development. Mouheb et al. [8] developed a UML profile that allows modeling security mechanisms as UML annotated aspect models to be woven automatically into a UML design model. Horcas et al. [9] propose an Aspect-Oriented Modeling (AOM) approach to weaving customized security models into an application using the Common Variability Language (CVL) and the Atlas Transformation Language (ATL). These works have left the Verification & Validation activity for future work. In addition, conflicts between the design and other architectural attributes may occur during this task after the weaving. In [10], Georg et al. proposed an approach for modeling security mechanisms and attacks as aspects using UML. To prove that the integration is correct, they use model verification on the application composed of the attack model and the security mechanisms.

The authors in [2] have reviewed security pattern specifications and usage. The conclusion indicates that there is minimal attention given to pattern integration. Peldszus et al. [11] introduce the GRaViTY approach which offers tools to align various artifacts generated during the model-driven development process, along with mechanisms for defining, applying, and reusing security requirements. Consequently, while the GRaViTY approach shows promise in supporting model-driven development, its limited scope and lack of empirical evidence may restrict its applicability and effectiveness in real-world scenarios. The work of [12] addresses the critical challenge of ensuring security in online service-oriented systems through a pattern-oriented approach. While numerous security design patterns exist, their integration remains a less explored area, motivating the proposed methodology. By utilizing the algebraic specification language SOFIA (Service-Oriented Formalism In Algebras) and translating specifications into the Alloy formalism, the work introduces a systematic approach to verify the validity and correctness of security design pattern compositions. The development of a tool support facilitates automated verification, demonstrated through a crowdfunding application case study. Despite advancements, existing works lack focus on proving functional correctness of pattern compositions, presenting an opportunity for future research. Additionally, the work highlights the potential of algebraic specifications for automated testing, suggesting avenues for further experimentation and extension of the proposed methodology. The approach faces scalability challenges with complex systems and struggles with adaptability to new security threats. Practicality concerns in environments that favor rapid development may also arise, questioning the method's efficiency in fast-paced settings. These factors highlight the need for a balance between rigorous security verification processes and the dynamic requirements of modern software development cycles. In [13], the authors introduce a groundbreaking aspect-oriented models-centered security framework aimed at addressing security challenges intrinsic to intelligent systems. The framework's components are carefully crafted based on identified threats specific to intelligent systems, with each element depicted through Unified Modeling Language (UML) diagrams. Other works like Armoush and al. [14] have focused on the integration of security patterns in the design of safety-critical embedded systems.

III. MBSPI APPROACH

In this section, the MBSPI approach is presented. This process is based on the approach done in [15] and previous work [16]–[18] which provides the first solution for design pattern integration in the context of object-oriented applications. Here, we go a step further. The validation of security properties and constraints has been formalized with OCL[1], which is a standard developed by the Object Management Group (OMG).

Fig. 1 depicts the MBSPI process which consists of five phases: Preparation, Elicitation, Context Validation, Merge, and Verification & Validation. The phases are described after presenting the pattern integration artifacts consumed and produced by each phase.

A. Definitions

The process interacts with the following artifacts:

- **Security Pattern** represents a modular part of a system that encapsulates a solution of a recurrent security problem in a specific context. The pattern and its constituents are developed by an security expert.
- **Application Diagram** is the representation of the software architecture of the application.
- **Pattern representation artifact** is the security pattern solution. It represents the architectural solution of the pattern. It consists of a number of software components: *participants* and *security mechanisms*.
- **Participants (roles)** represent generic components of a *security pattern solution*. They are the roles potentially played by components of the application.
- **Security mechanisms** are software components part of the *security pattern solution*. They provide primitive security functions (e.g., encryption, signing). A library of these functions is provided in order to allow security pattern solution modeling.
- **Preconditions** are the constraints that the application must verify in order for the integration to work.
- **Postconditions** are pattern security properties that the application verifies after the integration of the pattern.
- **Casting Diagram** consists of the *application diagram*, the *pattern representation artifact* diagram and the *bindings* between components of the two diagrams. The

[1]https://www.omg.org/spec/OCL/

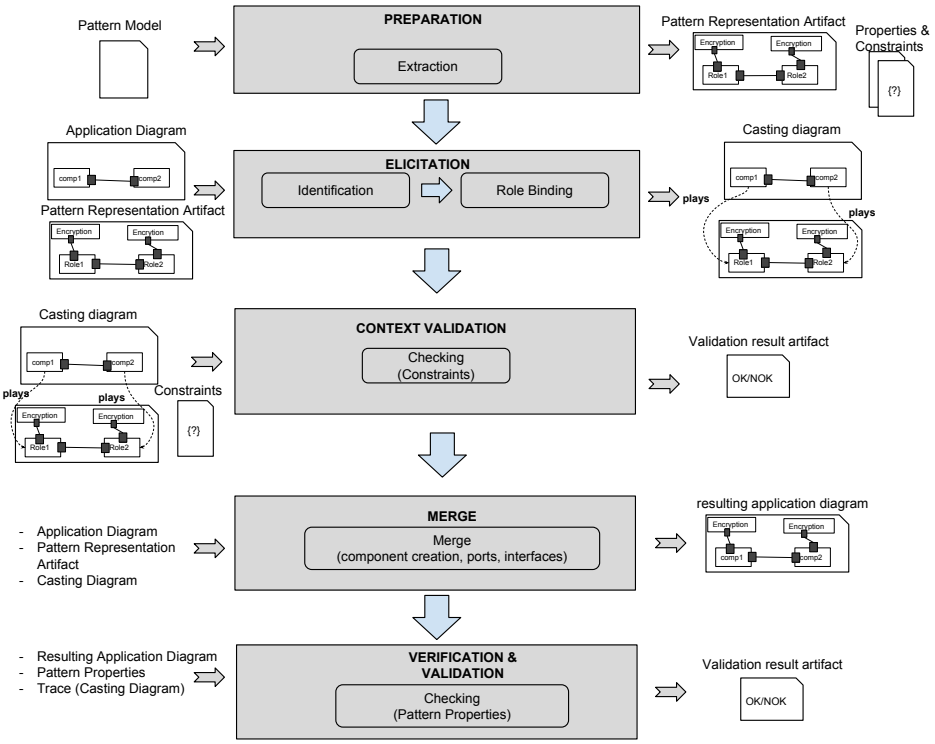

Fig. 1. MBSPI Process

bindings constitute mappings to identify the application components that play roles (participants) in the security pattern solution.

- **Resulting Application Diagram** is the final software architecture diagram once the pattern has been integrated into the application.

B. Hypothesis

Below are a set of assumptions that were used to construct the integration process:

- *Hypothesis 1: Security feature incorporation.* The pattern integration process is only intended to incorporate security features. We verify that the application diagram contains the necessary functional features before starting the integration.
  For example, in order to integrate a Secure Communication Pattern between a Client and Server components:
  - There must be communication.
  - Client and Server interfaces must have *Send()* and *Receive()* interfaces.
- *Hypothesis 2: Pattern properties relevant to messages.* The focus is particularly on the following pattern properties intended for transmitted messages:
  - **Property 1.** Confidentiality of messages.
  - **Property 2.** Integrity of messages.
  - **Property 3.** Authenticity of messages.
- *Hypothesis 3: Pattern constraints.* Preconditions are conditions that the artifacts must meet before going further:
  - **Precondition 1.** All pattern participants should be bound (one participant per component).
  - **Precondition 2.** All communications in the pattern should exist in the application.

C. Phase 1 (Preparation)

The purpose of this initial phase is to distill essential artifacts from a security pattern for use in the subsequent phase, specifically: *pattern representation artifact, properties (postconditions), constraints (preconditions)*. This phase serves as the preparatory step for the pattern integration process, where the critical inputs required for the process are identified and readied, effectively setting the stage for the successful application of the security pattern.

D. Phase 2 (Elicitation)

This phase aims to identify where and how the pattern will be applied. This Elicitation phase involves two key activities: *identification* and *role binding*. Initially, the architect pinpoints the software components within the application diagram that are suitable for the application of the pattern representation artifact. This step involves identifying components that are candidates to fulfill roles within the pattern representation artifacts. Following the identification of these components, the architect assigns them to their respective *roles* within the patterns, as depicted in Fig. 1 through the use of dashed arrows marked with *plays* stereotypes.

E. Phase 3 (Context Validation)

The objective of this phase is to ascertain whether the application satisfies the specified preconditions of the pattern. This phase takes as input the *Casting Diagram* and pattern *Preconditions* and provides as output a Result Artifact (that contains, for each constraint, the result of the checking). It is imperative that all preconditions are met before proceeding to the subsequent phase of the integration process, ensuring that the foundational requirements for security pattern integration are thoroughly validated.

F. Phase 4 (Merge)

The aim of this phase is to correctly integrate the pattern using the *Casting Diagram*. The Merge phase involves the integration of *Security Mechanisms*, ports, interfaces, and communication features from the Pattern Representation Artifact into the application's architectural diagram. This integration culminates in a revised version of the application, which is considered a candidate for enhanced security. This phase is crucial for ensuring that the security improvements are accurately and effectively incorporated into the application's design, aiming to elevate the overall security posture of the application. Let A be an application, P a pattern representation artifact and C a casting diagram containing a set of bindings bi. The resulting application obtained by integration process RA is defined by the algorithm in Listing 1.

```
1  Algorithm Merge
2
3  Input: A, P, C.
4  Output: RA.
5
6  RA:= duplicate(A)
7  C':= duplicate(C)
8  for each bi in C'
9    component1 = bi.component
10   patternParticipant1 = bi.patternParticipant
11
12   for each pPort in patternParticipant1
13    if pPort.communication.prts.component->includes(
          SecurityMechanism)
14       Prt1 = RA.duplicatePort(pPort,component1)
15       securityMechanism = RA.duplicateComponent(Port.
             securityMechanism)
16       RA.CreateCommunication(Prt1,securityMechanism.
             port)
17    else
18     patternPartipant2 = Port.communication.ports.component
         ->select(PP | PP != patternParticipant1)
19     component2 = patternParticipant.binding.component
20     Prt1 = RA.ports->select(prt| prt.owner = component1 and
          Prt1.communication.connects(component1,component2)
         )
21
22     for each pOperation in Port.interface.operations
23      if Prt1.interface.operations->includes(pOperation) ==
            false
24       RA.addOperation(pOperation, Prt1.interface)
25      endif
26     endfor
27
28    endif
29   endfor
30 endfor
```

Listing 1. Merge algorithm

First, the application diagram A and casting diagram C are duplicated and named RA and C' respectively. The set of bindings in C' are parsed. For each binding bi, $patternParticipant1$ and $component1$ are the pattern participant and the application component bound by bi respectively. In addition, ports owned by $patternParticipant1$ are looked up. Afterward, security mechanisms are deployed. For each port $pPort$, if $pPort$ connects $patternParticipant1$ to a security mechanism, then it is duplicated, added to $component1$ in RA, and named $Prt1$. The security mechanism is duplicated, added to RA, and named $securityMechanism$. A communication is created between $Prt1$ and a $securityMechanism$ port. Finally, the necessary operations are added to the interfaces of the application components in order to correctly call the operations of the security mechanisms. If $pPort$ does not connect $patternParticipant1$ to a security mechanism, then it is connected to another pattern participant that we name $patternParticipant2$. We name $component2$ the application component bound to $patternParticipant2$. According to the assumptions, there must be communication between application components (this is a precondition). In this case, $component1$ is connected to $component2$ via port $Prt1$. The operations of $pPort$ interface are looked up. For each operation $pOperation$ in $pPort$ interface, if $pOperation$ does not exist in $Prt1$ interface, $pOperation$ is added to the operations of $Prt1$ interface. After the merge phase, the resulting application verifies the *pattern postconditions*.

G. Phase 5 (Verification & Validation)

At this stage, the new application verifies the post- conditions. At each change (e.g., ad-hoc tailoring or the integration of another pattern), the application is validated against the postconditions. A dedicated checking module is responsible for ensuring the application adheres to these postconditions, which reflect the essential security properties of the pattern. Modifications to the application, whether due to specific customizations or the addition of another security pattern, are scrutinized under this process to ensure the security enhancements are properly maintained.

The application's compliance with pattern postconditions is assessed by ensuring: (1) the inclusion of necessary security mechanisms, like encryption, to uphold the pattern's security attributes within the application, and (2) the correct application of these mechanisms, such as encrypting messages, to safeguard data integrity and confidentiality during transmission. This process confirms both the presence and proper implementation of security measures as stipulated by the security pattern.

IV. MDE Framework

MDE is used to support the aforementioned approach, focusing on software architecture and security patterns. For architectural modeling, we used a Domain Modeling Specific Language (DSML) based on UML modeling language

to describe software architecture using the component-port-connector fashion. For specifying and analyzing security properties, OCL is utilized, allowing for precise definition and verification of security attributes within the system.

Modeling the architecture.: In the context of Component-Based Development (CBD), the "ComponentUML" UML profile was developed to facilitate application modeling. This necessity for a specialized profile emerged during the formalization of OCL expressions, where the application of standard UML led to complexities due to the inclusion of irrelevant concepts. To streamline this process, the "ComponentUML" profile was crafted, focusing on simplification. It is built upon key UML constructs, specifically StructuredClassifiers, Messages, and Deployments, to tailor the modeling experience to CBD's specific needs.

Working example: metamodel instantiation: Fig. 2 shows the software architecture of a three-tier web application. The structure of the architecture is delineated into three fundamental types of components: *Page*, *Webapp*, and *Database*. Each of these components is interconnected through specific ports, interfaces, data types, and messages. For example, a Page-type component, referred to as *Webapp*, leverages a "Port Client Server" for communication with the *Webapp* component, which is classified as a Webapp-type. In response, the *Webapp* component utilizes a "Port Server Client" for reciprocal communication. Annotations in blue delineate various messages: "m1" describes the request dispatched to the application, "m2" illustrates the application's response, "m3" signifies the request made to the database, and "m4" represents the response received from the database. From a deployment standpoint, the architecture is supported by three nodes, which serve distinct functions: the *Browser* node, which hosts the *webpage*; the *Server* node, which is accessible via the Internet and hosts the *Webapp*; and a backend node dedicated to database hosting.

Modeling security patterns: We developed a UML profile called *SepmUML* using UML notations. *SepmUML* contains the necessary stereotypes for modeling a security pattern in UML environments. The solution of the security pattern is modeled using ComponentUML. In addition pattern integration-related concepts are specified. The specification of the UML profile and the pattern specification is out of scope in this paper and is detailed in [19]. In this process, different stakeholders interact in order to specify the pattern and its constituents including the pattern solution and the OCL constraints. Five main security mechanism categories of SepmUML are considered: Authenticity, Authorisation, Authentication, Cryptography, Monitoring, and Filtering. Derived from security requirements, a customized security pattern solution can be built up from a combination of these security mechanisms categories.

Working example : SSL pattern: Fig. 3 shows the solution of the SSL (Secure Sockets Layer) pattern. It has two pattern participants *clientParticipant* and *serverParticipant*. It contains the following security mechanisms. The *protocol controller* is the main component that manages the other security features to facilitate the execution of the *SSL Handshake* and *SSL record* protocols. *The authenticator* validates the identity of either the client or server by verifying their certificates. The *key exchange* component calculates the key exchange for the client, which is essential for encrypting communications. While the *encryptor* applies encryption to outgoing messages utilizing a specific key and the *decryptor* utilizes a key to decrypt incoming messages. The *Signer* generates a digital signature for each message to ensure its authenticity and integrity, which accompanies the transmitted message. The *verifier* confirms the authenticity and integrity of a message by examining the digital signature that comes with it. The description of the properties is encapsulated within a comment annotated by the stereotype *PropertySpecification*, encompassing the attributes *confidentiality*, *integrity*, and *authenticity*. As shown at the bottom of Fig. 3, the pattern provides the following properties: confidentiality of messages m_1 and m_2 **(Property 1)**, integrity of messages m_1 and m_2 **(Property 2)**, authenticity of messages m_1 and m_2 **(Property 3)**. The preconditions are: All pattern participants should be bound (one participant per component) **(Precondition 1)**. Additionally, all pattern-defined communications must be present within the application, identified as **(Precondition 2)**.

Casting diagrams and preconditions: Fig. 4 shows the following bindings between the SSL patterns and components of the web application where "webpage" and "webapp" *play* role "clientParticipant" and "serverParticipant" respectively. The casting diagram is validated against *Precondition 1 and 2*. The validation of the castings against the preconditions is done using OCL invariants. Listing. 2 and Listing. 3 validate *Precondition 1 and 2* respectively. The two preconditions have been already explained previously and recalled as comments at the beginning of the listings. In this case, the preconditions are valid so we move to the next phase.

```
1  //Precondition 1 : All pattern participants should be bound
       (one participant per component)
2  Context Castings
3  self.play->select(p1,p2 |
4
5  (p1.participant = p2.participant
6  implies
7   p1.component = p2. participant)
8
9  and
10
11 (p1.component = p2. participant
12 implies
13 p1.participant = p2.participant)
14
15 and
16 p1.participant.structureContainter.participants->forAll(
       participant | self.play->exists(p_ | p_.participant =
       participant)))
```

Listing 2. Precondition 1: All pattern participants should be bound (one participant per component)

```
1
2  //Precondition 2: all communications in the pattern should
       exist in the application
```

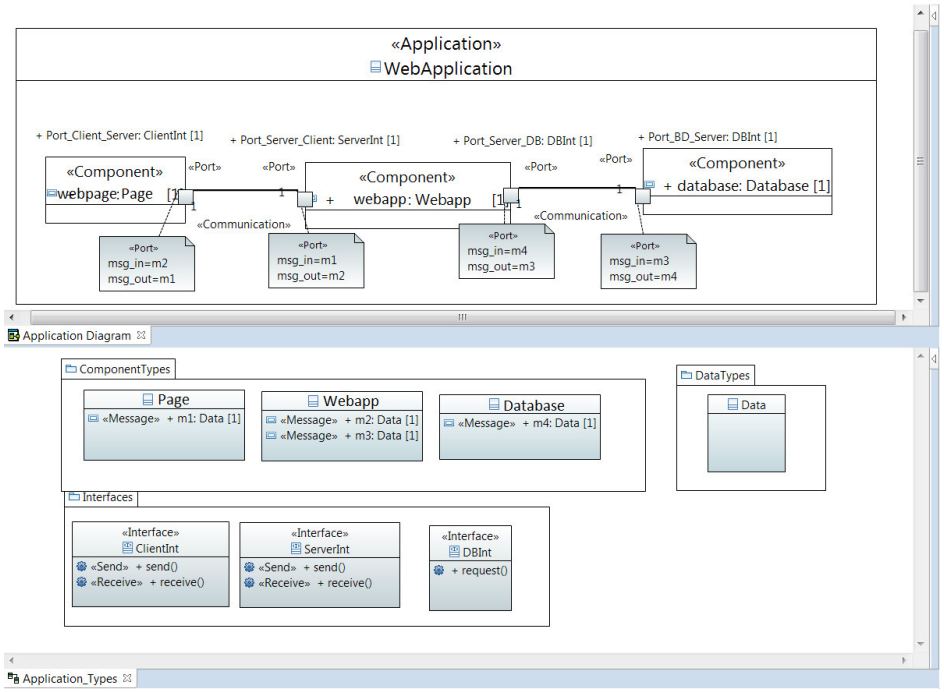

Fig. 2. Architecture Model and Component Types of a Web Application [18]

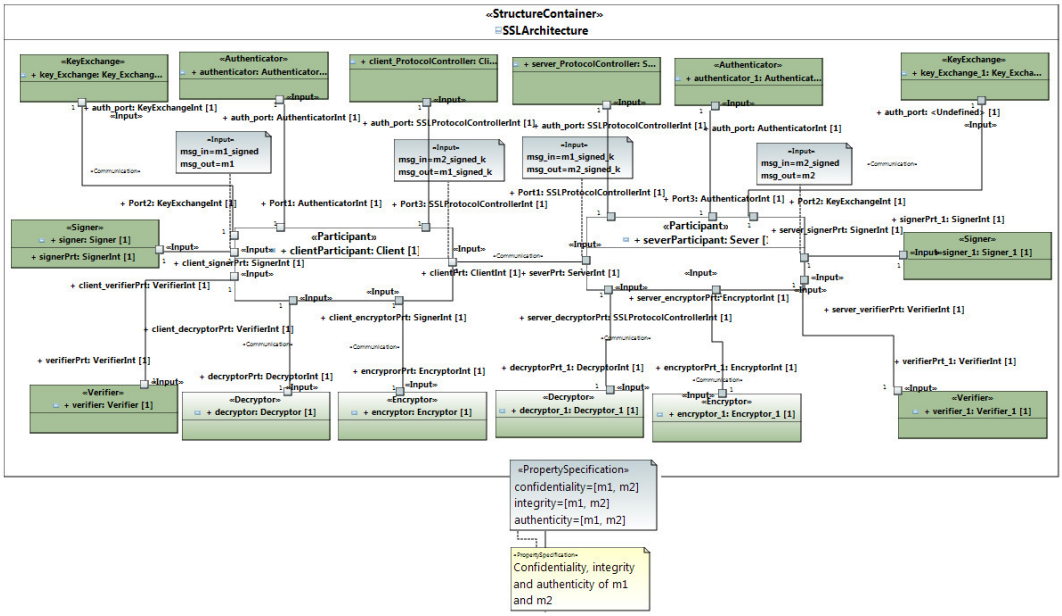

Fig. 3. SSL Pattern Solution

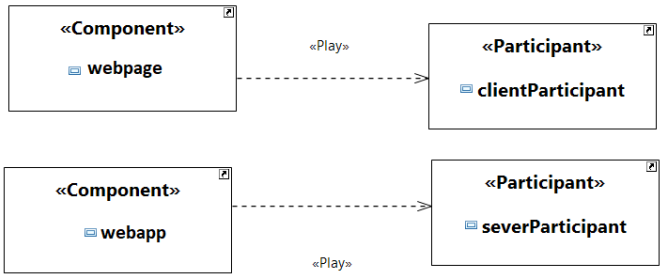

Fig. 4. Casting Diagram for Application Communication Using SSL Security Pattern

```
Context Castings
self.play->forAll(p1,p2 |
 (p1._'<>'(p2) and p1.participant.ports->exists(p1_in | p2.
      participant.ports->exists( p2_in | p1_in.communication
       = p2_in.communication)))
implies
 p1.component.ports->exists(p1_in | p2.component.ports->
      exists( p2_in | p1_in.communication = p2_in.
      communication))
)
```

Listing 3. Precondition 2: all communications in the pattern should exist in the application

Merge: In this phase, a transformation from Model to Model is executed, utilizing the procedure outlined in Listing 1 written with QVT (Query, View, and Transformation)[2]. This transformation process inputs the application depicted in Fig. 2 along with the castings shown in Fig. 4. The result of the Merge is a refreshed application model illustrated in Fig. 5. Furthermore, the initial application's types and interfaces undergo modifications with the addition of new attributes and operations, highlighted in green. Additionally, new interfaces pertinent to security mechanisms have been introduced.

Context checking: At this phase, the new application undergoes a verification process to ensure it meets the post-conditions. Whenever a modification occurs, such as ad-hoc adjustments or the amalgamation of an additional pattern, the application's conformity with the postconditions is reassessed. This evaluation employs the OCL invariant as detailed in Listing. 4. If the postconditions still hold, the change is accepted and committed. Else, the change is dismissed. The validation process involves the application being checked against *Property 1, 2, and 3* through the confirmation of the following criteria:

- There must be one *encryptor* and *signer* mechanisms for each component sending messages m_1 and m_2.
- Messages m_1 and m_2 must be encrypted before being sent.

```
//Postconditions validation
Context Application
self.base_class.ownedAttributes->select(c | c.isoclAsKind(
     Comment))->select(c |
c.isStereotypeApplied(PropertySpecification))->forAll( p |

p.confidenlity->forAll( m_confidential |
// There exists a component that produces m_confidential
     and an ecryptor connected to this component that
     encrypts m_confidential
self.components->exists( c1|
c1.ports.msg_out = mconfidential
and
self.components->exists( enc|
enc.isOclKindOf(Encryptor)
and
c1.ports->exists(c1_in | enc.ports->exists(enc_in | c1_in.
     communication =
enc_in.communciation )))))
and
p.integrity->forAll( m_integrity |
// There exists a component that produces m_integrity and a
     signer connected to this component that signs
     m_integrity
self.components->exists( c1|
c1.ports.msg_out = m_integrity
and
self.components->exists( signer|
signer.isOclKindOf(Signer)
and
c1.ports->exists(c1_in | signer.ports->exists(signer_in |
     c1_in.communication =
enc_in.communication ))))))
```

Listing 4. OCL Queries for pattern property verification

V. Tool Support

The proposed toolchain is designed to support the proposed metamodels (ComponentUML and SepmUML) and Model-To-Model transformations. To support our approach, tools must fulfill the following key requirements:

- Allow the creation of a custom UML profile and UML models.
- Support the implementation of a repository to store pattern models and the related model libraries for classification and relationships.
- Support the access to the repository. Create views on the repository according to its APIs, its organization, and the needs of the targeted system engineering process.
- Enable transformations of the pattern models from the repository format into the target-modeling environment.
- Enable the creation of System of Patterns configuration models in the target-modeling environment.
- Enable the integration of patterns imported from the model repository into application models.

In our case, the following support tools have been chosen:

- UML modeling environment: Papyrus[3] (Existing)
- Model Repository : SEMCOMDT[4] (SEMCO Model Development Tools, IRIT's editor and platform plugins) is used to support pattern repository (Existing).
- Selection, Instantiation, and Integration of Pattern Models: Semco4Papyrus (Implemented)

VI. Case Study: SCADA System

This section evaluates the applicability and efficacy of our proposed methods through the study of a SCADA system. SCADA systems diverge significantly from conventional IT

[2]https://www.omg.org/spec/QVT

[3]https://eclipse.org/papyrus/

[4]http://www.semcomdt.org

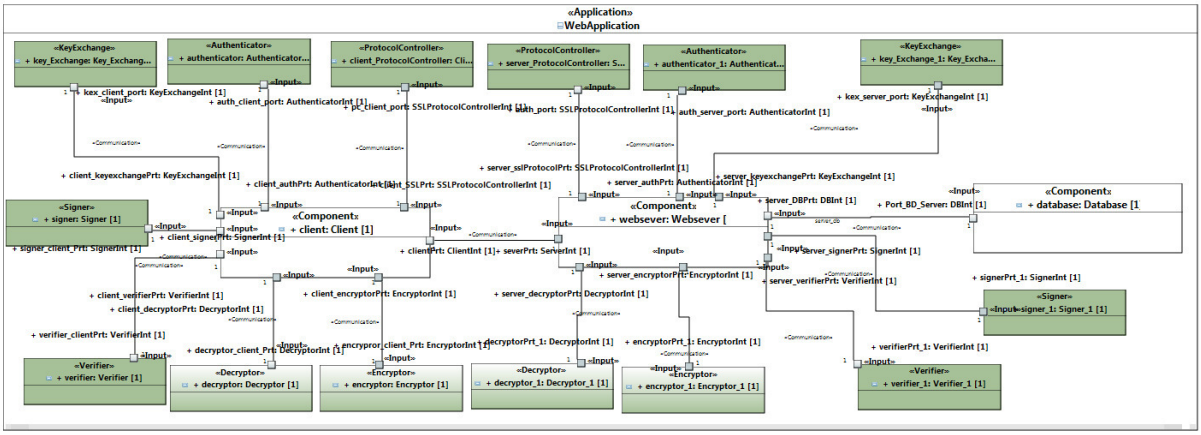

Fig. 5. New Application diagram

frameworks, including web applications, showcasing particular and rigorous security demands. The purpose of this evaluation is to illustrate the capability of our suggested approaches in meeting the sophisticated security challenges that are fundamental to these vital infrastructure components.

Description and modeling: In our study, we explore a simplified SCADA system tailored for smart grid applications. This system is designed to perform essential functions such as: (1) Perform control, (2) Poll Data, (3) System Start-up/shutdown, (4) Adjust Parameter Settings, (5) Log Field Data, (6) Archive Data, (7) Trigger Alarm, (8) Perform Trending (e.g., Select Parameters, Display Parameters, Zooming and Scrolling). Fig. 6 depicts the software architecture model.

A set of patterns is selected from the model repository and then instantiated in the modeling environment Papyrus[5] (see Fig. 7). The security patterns are:

- *Secure Communication* that ensures that data passing across a network is secure. It can be refined by two alternative patterns: SSL and IPSec.
- *Firewall* that restricts access to the internal network. It can be refined by the following alternative patterns: Packet Firewall and Stateful Firewall.
- *Intrusion Detection System (IDS)* in order to stop malicious payloads.
- *Authorization* that ensures that only authorized users are allowed access.
- *RBAC (Role-Based Access Control)* that allows the craetion of a set of roles. Each role has a set of defined rights.
- *Logger and Auditor*. that allows the logging of actions.

Amongst the different Patterns, two have been selected: SSL and Packet Firewall patterns and integrated into the SCADA system software architecture. The integration aims at protecting: (1) the communication between the SCADA server and PLCs (Programmable Logic Controller) against information disclosure and spoofing (e.g., Man-In-The middle attacks), (2) the SCADA server and PLCs against Denial of Service attacks.

[5]https://eclipse.org/papyrus/

Assessment: For the assessment, we use MBTA [18] which is a framework for detecting threats based on OCL. The aim is to analyze the software architecture before and after pattern integration. The approach and threat categories are detailed in [18]. Fig. 8 presents a comparative analysis of the frequency of threats categorized by type, before and following the adoption of security patterns. The implementation of the SSL pattern eradicates threats associated with Man-In-The-Middle and Tampering (during transmission) by ensuring the confidentiality and integrity of communications between the SCADA server and the PLCs. However, vulnerabilities to Denial of Service and Injection attacks persist, affecting four software components: HMI (Human-Machine Interface), Trending, LogDisplay, and AlarmDisplay; due to their exposed public ports and the absence of Firewall and Authorization safeguards.

VII. Conclusion

In this paper, we have proposed an approach and tool support for integrating proper security patterns into software architectures, aligning with OMG (Object Management Group) standards such as UML, profile extension mechanism, and OCL. Our process utilizes merging and OCL verification strategies within an MDE based approach to facilitate the integration of security patterns. The challenge of pattern integration lies in seamlessly embedding all components of a pattern into an application, ensuring system integrity and quality are maintained, and affirming the enhancement of the system with the new properties introduced by the pattern. While the majority of research in this area concentrates on utilizing merging techniques for the incorporation of patterns into applications, our study emphasizes the Verification & Validation phase to validate the integration. Through the examination of a case study and the conducted assessment, we have pinpointed limitations in the current iteration of MBSPI. Our future directions include the proposition of pattern bindings to users, tailored according to the component types targeted for pattern application. During the integration phase, our focus has been predominantly on the structural aspects of architecture, as

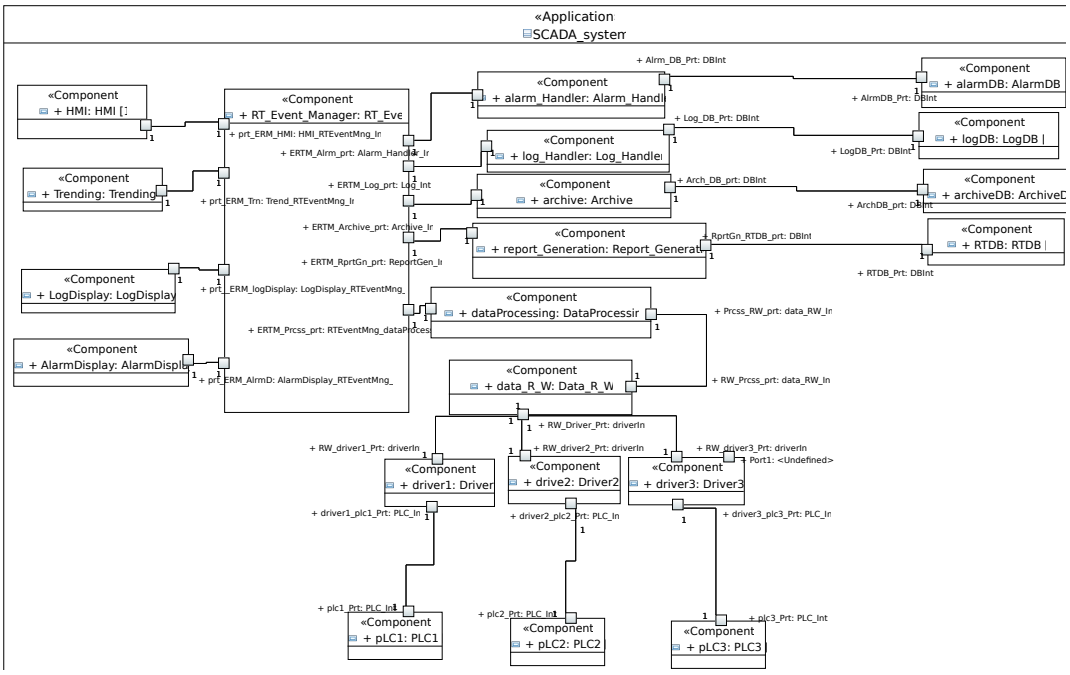

Fig. 6. SCADA software architecture model

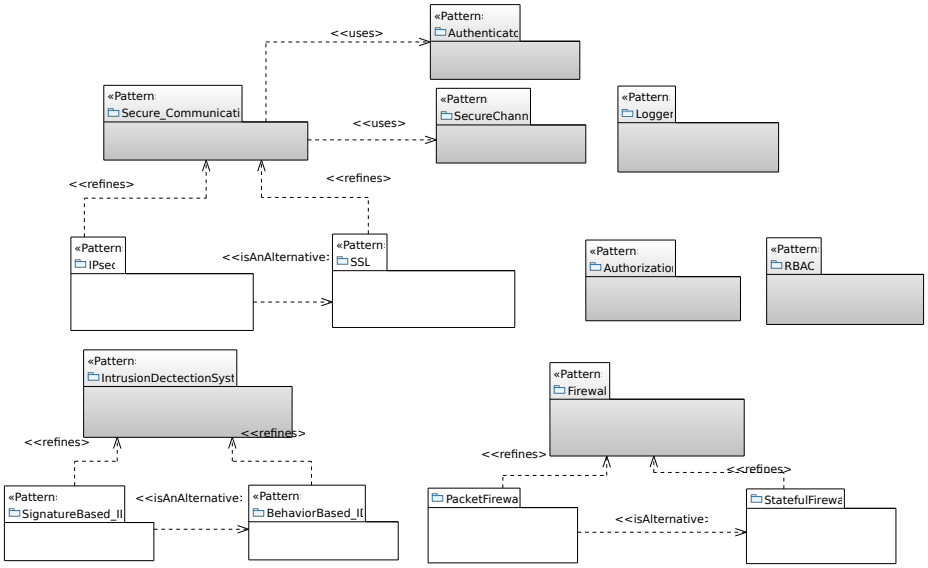

Fig. 7. Selected Security Patterns

depicted in UML composite diagrams. Moving forward, it is imperative to also account for behavioral considerations during the integration process. Currently, messages are treated as structural elements within our solution; however, to adequately represent the sequencing and interaction of messages, we propose the utilization of dedicated diagrams, such as the UML sequence diagram, which better captures message order and interaction dynamics.

REFERENCES

[1] E. Fernandez-Buglioni, *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[2] H. Washizaki, T. Xia, N. Kamata, Y. Fukazawa, H. Kanuka, T. Kato, M. Yoshino, T. Okubo, S. Ogata, H. Kaiya, *et al.*, "Systematic literature review of security pattern research.," *Information*, vol. 12, no. 1, pp. 2078–2489, 2021.

[3] D. Manolescu, W. Kozaczynski, A. Miller, and J. Hogg, "The Growing Divide in the Patterns World," *IEEE Software*, vol. 24, pp. 61–67, July 2007.

[4] D. Serrano, A. Mana, and A.-D. Sotirious, "Towards Precise and Certi-

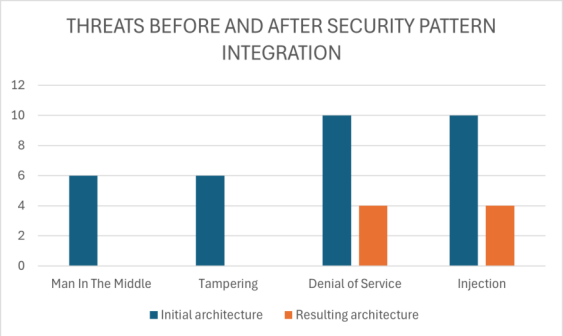

Fig. 8. Threats Counts Before and After Security Pattern Integration

fied Security Patterns," in *Proceedings of 2nd International Workshop on Secure systems methodologies using patterns (Spattern 2008)*, pp. 287–291, IEEE Computer Society, September 2008.

[5] J. F. Ruíz, M. Arjona, A. Maña, and N. Carstens, "Secure engineering and modelling of a metering devices system," in *2013 International Conference on Availability, Reliability and Security*, SecSE'13, pp. 418–427, IEEE, 2013.

[6] A. Maña, E. Damiani, S. Gürgens, and G. Spanoudakis, "Extensions to Pattern Formats for Cyber Physical Systems," in *Proceedings of the 31st Conference on Pattern Languages of Programs*, no. 15 in PLoP'14, pp. 15:1–15:8, ACM, 2014.

[7] P. H. Nguyen, K. Yskout, T. Heyman, J. Klein, R. Scandariato, and Y. L. Traon, "SoSPa: A system of Security design Patterns for systematically engineering secure systems," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 246–255, Sept. 2015.

[8] D. Mouheb, C. Talhi, M. Nouh, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi, "Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML," in *Software Engineering Research, Management and Applications*, no. 296 in Studies in Computational Intelligence, pp. 197–213, Springer Berlin Heidelberg, 2010.

[9] J. M. Horcas, M. Pinto, and L. Fuentes, "An Aspect-Oriented Model transformation to weave security using CVL," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 138–150, Jan. 2014.

[10] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. H. Houmb, "An aspect-oriented methodology for designing secure applications," *Information and Software Technology*, vol. 51, pp. 846–864, May 2009.

[11] S. Peldszus, "Model-driven development of evolving secure software systems," in *Combined Proceedings of the Workshops at Software Engineering 2020 Co-located with the German Software Engineering Conference 2020 (SE 2020)* (R. Hebig and R. Heinrich, eds.).

[12] X. Zheng, D. Liu, H. Zhu, and I. Bayley, "Pattern-based approach to modelling and verifying system security," in *15th IEEE International Conference on Service Oriented Systems Engineering (SOSE)*, pp. 92–102, 2020.

[13] H. A. Alhamad and M. M. Hassan, "Aspect-oriented models-based framework to secure intelligent systems," in *Proceedings of the 8th International Conference on Computer Technology Applications (ICCTA)*, pp. 249–262, 2022.

[14] A. Armoush, "Towards the integration of security and safety patterns in the design of safety-critical embedded systems," in *4th International Conference on Applied Automation and Industrial Diagnostics (ICAAID)*, vol. 1, pp. 1–6, 2022.

[15] B. Hamid, C. Percebois, and D. Gouteux, "A Methodology for Integration of Patterns with Validation Purpose," in *Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP)*, pp. 1–14, ACM, 2012.

[16] R. Abdallah, A. Motii, N. Yakymets, and A. Lanusse, "Using model driven engineering to support multi-paradigms security analysis," in *Model-Driven Engineering and Software Development: Third International Conference, MODELSWARD 2015, Angers, France, February 9-11, 2015, Revised Selected Papers 3*, pp. 278–292, Springer, 2015.

[17] A. Motii, B. Hamid, A. Lanusse, and J.-M. Bruel, "Towards the integration of security patterns in UML component-based applications," in *Joint Proceedings of the Second International Workshop on Patterns in Model Engineering and the Fifth International Workshop on the Verification of Model Transformation*, vol. 1693 of *PAME '16*, pp. 2–6, CEUR-WS.org, 2016.

[18] A. Motii, "Mbta: A model-based threat analysis approach for software architectures," in *42nd International Conference on Computer Safety, Reliability, and Security (SafeComp)*, pp. 121–134, 2023.

[19] A. Motii, B. Hamid, A. Lanusse, and J. M. Bruel, "Guiding the selection of security patterns for real-time systems," in *21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 155–164, 2016.