

# From Secure Business Process Modeling to Design-Level Security Verification

Qusai Ramadan\*, Mattia Salnitri<sup>†</sup>, Daniel Strüber\*, Jan Jürjens\*<sup>‡</sup> and Paolo Giorgini<sup>†</sup>

\*Institute for Software Technology IST, University of Koblenz-Landau, Koblenz, Germany

Email: {qramadan, strueber, juerjens}@uni-koblenz.de

<sup>†</sup>University of Trento, Trento, Italy

Email: {mattia.salnitri, paolo.giorgini}@unitn.it

<sup>‡</sup> Fraunhofer-Institute for Software and Systems Engineering ISST, Dortmund, Germany

**Abstract**—Tracing and integrating security requirements throughout the development process is a key challenge in security engineering. In socio-technical systems, security requirements for the organizational and technical aspects of a system are currently dealt with separately, giving rise to substantial misconceptions and errors. In this paper, we present a model-based security engineering framework for supporting the system design on the organizational and technical level. The key idea is to allow the involved experts to specify security requirements in the languages they are familiar with: business analysts use BPMN for procedural system descriptions; system developers use UML to design and implement the system architecture. Security requirements are captured via the language extensions SecBPMN2 and UMLsec. We provide a model transformation to bridge the conceptual gap between SecBPMN2 and UMLsec. Using UMLsec policies, various security properties of the resulting architecture can be verified. In a case study featuring an air traffic management system, we show how our framework can be practically applied.

## I. INTRODUCTION

The vast majority of today’s software systems are part of Socio-Technical Systems (STSs) that involve a rich interplay of organizational (humans and organizations) and technical (software and hardware) components to accomplish shared objectives [1]. Examples include air traffic management (ATM), smart cities, and healthcare systems. STSs open up a new class of security challenges. Since STSs are decentralized and their components are autonomous and loosely controllable, a security issue in a single component may affect other components, leading to unpleasant consequences such as privacy violations, law infringements, and safety risks. For instance, in an ATM system, unauthorized modification of the flight plan of a flying plane can threaten the safety of passengers, with severe consequences for the airline and airport companies.

For the effective engineering of a secure STS, it is important to consider both organizational and technical security requirements right from the start of the development process. Therefore, various approaches have been proposed to incorporate security engineering into the early stages of the system development life cycle.

A number of *BPMN-based approaches* [2], [3] rely on business process modeling for organizational security requirements as early as during the design of the business processes for the target STS. These approaches abstract from technical details

to allow the specification of high-level security requirements by non-technical stakeholders, such as business analysts [4].

Moreover, a number of *UML-based approaches* [5], [6] allow the system developers to design a secure architectural model for the target system. This model is enriched with low-level technical details to support the validation against predefined technical security policies. The system architecture model is a cornerstone for further development stages, such as generating code for the implementation [5], [7].

Model-based approaches for the engineering of secure systems are widely believed to be useful [8]. However, since the existing BPMN- and UML-based approaches address security in distinct development phases and from the perspectives of different stakeholders, they deal with security requirements separately; therefore, an alignment of security requirements across the different phases is not guaranteed [9]. Since building secure systems is a sensitive task, it is important to manage security consistently, so that the introduction of vulnerabilities during the development process is avoided.

A main source of vulnerabilities are misunderstandings between expert stakeholders, as triggered by their implicit knowledge about terminology. For instance, one of the most common uses of *swimlanes* is to express internal roles in an organization, while the BPMN 2.0 standard leaves their semantics undefined [10]. Without this implicit knowledge about the usage of swimlanes, system developers may forgo the use of an appropriate role enforcement mechanism (e.g., role-based access control), resulting in security loopholes. Worse, such loopholes may be notoriously hard to detect due to lack of a traceability mechanism for security requirements across the different phases. Support for an integrated management of organizational and technical security requirements throughout the entire development process is generally lacking.

To address these shortcomings, in this paper, we propose a framework for designing secure STSs by integrating existing BPMN-based and UML-based security engineering approaches. In particular, our framework suggests to iteratively (i) model organizational requirements and security requirements using SecBPMN2 [3], (ii) transform the SecBPMN2 model to a preliminary architectural model enriched with security policies using UMLsec [5], (iii) refine the generated UMLsec model manually with additional design decisions, and

(iv) verify the resulting UMLsec model against their contained security policies by using an automated tool, called CARiSMA [11]. Specifically, our contributions are:

- a *semi-automated process* for enforcing an integrated security management throughout the development process (Sect. 3),
- a *model transformation* supporting the translation of security-annotated business models to secure architectural models while establishing traceability (Sect. 4), and
- a *case study* featuring an ATM system, showing how our framework can be used to establish an integrated management and traceability of security requirements (Sect. 5).

The novelty of our framework is that we automatically establish traceability between high-level security requirements and verifiable technical security policies. Doing so, we integrate the views of business analysts and system developers, the two main kinds of expert stakeholders during the development of STSs. Earlier automated transformation approaches used UML as sole modeling language throughout the entire process [12], thereby leaving the role of business analysts unaddressed, or focused on representing security “*at the business analyst views*” ([13], p.2), leaving technical security concerns and the verification of security aspects to future work [13]. These and other, non-automated, transformation approaches are discussed in Sect. VI.

## II. FOUNDATIONS

In this section, we briefly describe necessary foundations for modeling and validating security requirements and policies, focusing on the approaches SecBPMN2 and UMLsec, which we illustrate with example models from our case study.

**BPMN-based security engineering** allows business analysts to express and reason with organizational security requirements using the Business Process Modeling Notation (BPMN, [10]). These approaches hide technical details of STSs, while permitting non-technical stakeholders to specify high-level security requirements in an intuitive way [4].

We selected SecBPMN2 [3] as a basis for our framework due to its expressiveness and the ability to model both business processes and security requirements. To specify security requirements in business process models, SecBPMN2 provides

11 security annotations, such as *Accountability*, *Confidentiality*, and *Integrity*, that can be added and linked to BPMN 2.0 elements such as *tasks*, *data objects*, and *message flows*. A number of alternative approaches have been proposed in the literature where specific annotations are introduced to extend BPMN with security aspects [2], [14], [15]. However, these approaches are not designed to express security requirements [14], or they permit to represent only a restricted set of security aspects [2], [15]. More details about SecBPMN2 and its expressiveness can be found in [3].

*Example* Fig. 1 shows a SecBPMN2 diagram representing a business process for flight plan negotiation in an air traffic management system. Executors of a business process are represented by *Pools* such as *Airplane* and *Local authority*. Communications between pools are represented by *message flows*; the content of such communications are *messages*: *Notify local authority* sends the message *Flight plan* to *Local authority*. Atomic activities are represented with *tasks*, for example *Take off*. Events are represented with circles. *Start events* and *End events* mark the initial and terminal points of business processes. *Catch events* represent points in a business process where an event needs to happen, for example *20 minutes before new aerospace*. Gateways are used to specify deviations of the execution sequence: *exclusive gateway* specify decisions points: the gateway *Flight plan created?* allows the upper or lower branch to be executed, depending on whether the question is answered *Yes* or *No*.

Security concepts are represented with orange solid circles. In particular, *Integrity* and *Confidentiality* are associated to the message flow meaning respectively that the content of the message is preserved and it will not be accessed by unauthorized users. *Accountability* is associated to *Check flight plan* meaning that any user who will mis-execute such activity will be traced.

**UML-based security engineering** allows system developers to express and reason about security policies using the Unified Modeling Language (UML, [16]). The idea is to first construct a security-annotated architecture model of a system. In a second step, the implementation is derived from the model, either automatically or manually.

Many UML-based security approaches have been proposed in the literature such as misuse cases [17], mal-activity [18], UMLsec [5], SecureUML [6], SECTET [19]. Most of these

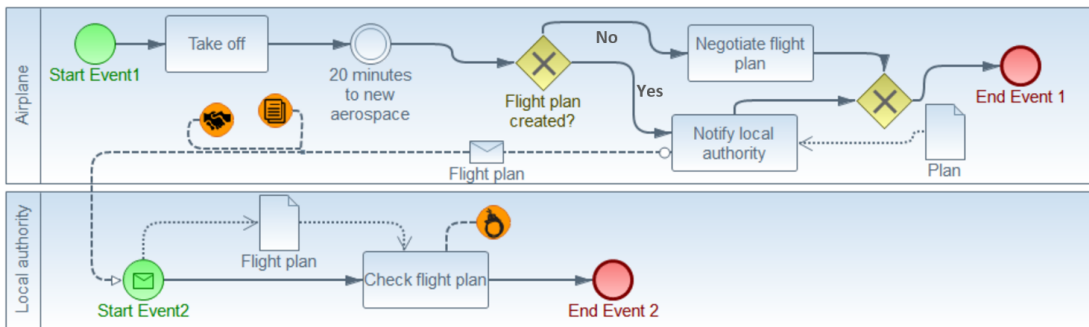


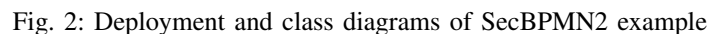
Fig. 1: Example of a SecBPMN2 diagram.

Stereotype	$Threat_{default}(S)$	$Threat_{insider}(S)$
«Internet»	{ delete, read, insert }	{ delete, read, insert }
«encrypted»	{ delete }	{ delete, read, insert }
«LAN»	$\emptyset$	{ delete, read, insert }
«wire»	$\emptyset$	{ delete, read, insert }

UMLsec [5] is a UML profile that can be used to enrich UML diagrams with security-relevant information. This paper focuses on deployment and class diagrams, which are typically used to define a system architecture. UMLsec extends UML with security-specific *stereotypes* and *tags*, that, when attached to UML element, add three kinds of security-relevant information: (i) assumptions on the physical environment of the system, such as «*Internet*» or «*encrypted*», (ii) security requirements on the logical structure of the system (such as «*secrecy*», «*integrity*») or on security-critical parts (such as «*critical*»), (iii) security policies that system parts are supposed to obey, such as «*secure links*» to ensure that security requirements on the communication are met by the physical layer, «*secure dependency*» to ensure that dependent parts in the architecture model preserve the security requirements relevant to the part they depend on, and «*abac*» to define central elements of the Role Attribute-Based Access Control (RABAC) [20] mechanism such as roles and permissions and verify them against the designed architecture. UMLsec has shown its usefulness in several industrial applications [21], [22], [23], in-house training courses exist at several companies.

The lower part of Fig. 2 is a class diagram annotated with «*abac*» and «*secure dependency*» policies along with their related annotations. In the «*secure dependency*» policy, the «*critical*» stereotype labels classes with sensitive data and operations, and the associated tags {*secrecy*} and {*integrity*} specify security requirements on these data and operations. More specifically, each class specifies the *stipulated* requirements of its own members, and the *fulfilled* requirements of other classes' members. Verifying «*secure dependency*» entails checking whether all requirements stipulated by a class are fulfilled by all dependent classes, which can be done automatically by the CARiSMA tool [11]. UMLsec implements the RABAC access control model via the policy «*abac*», which uses two tags called {*role*} and {*right*} to assign roles to subjects and rights to roles, respectively. Operations in need of an access restriction can be annotated with the «*abacRequire*» stereotype along with its associated {*right*} tag. An account and a formal foundation of the UMLsec stereotypes and tags is given in [5]; the RABAC extension was introduced in [24].

Integrating and tracing security requirements from business process models to the system architecture is a complicated



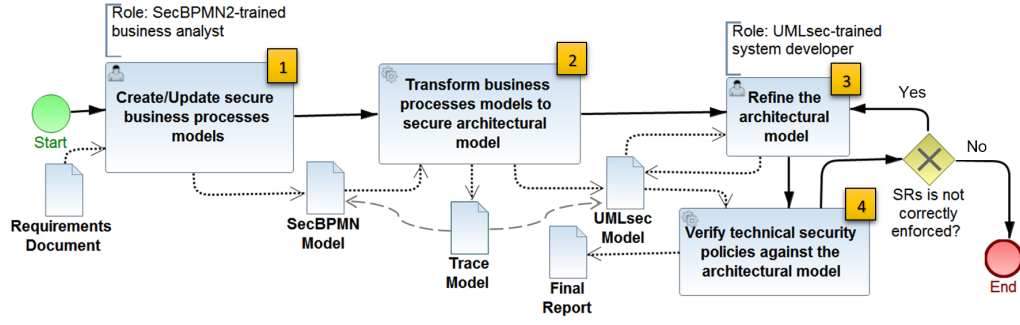


Fig. 3: The process proposed in this paper.

task. Business processes are mainly about behavior, tasks, and flows, which specify how a system achieves its goals, while architectural deployment and class diagrams focus on structural elements such as components, classes, and operations, which specify how the system should be built to achieve its goals. Therefore, it is not possible to automatically specify all the architectural details based on a given business process. For example, while a task in the business process can be transformed to an operation in the architectural model, a one-to-one mapping is not always possible, as it might be preferable to represent the task by a set of operations rather than one. For this reason, our framework proposes the semi-automated process shown in Fig. 3.

**Roles:** At a minimum, our process involves a team of *SecBPMN2-trained business analysts* and *UMLsec-trained system developers*. Our assumptions on the skills of these involved stakeholders are light-weight, as they do not have to be security experts. Still, to ensure correct use of SecBPMN2 and UMLsec, some additional instruction on top of their regular training is appropriate.

With these assumptions, our aim is to address the common situation in which security experts are absent [25]. In this situation, we aim to make the software more secure than it would be when ignoring security from the start. However, even in presence of security experts—which is clearly the preferable situation—, our process can still be helpful, as the involved expert and non-expert stakeholders may benefit from the traceability of security requirements across development phases.

**Input/Output:** As input, the process receives a requirements document containing organizational, technical, and security requirements. During the requirements elicitation, the business analysts produce this document in interaction with the customer. The output is a secure system architecture and a verification report, showing the results of validating the architectural model against the UMLsec security policies.

**Phase 1.** In this phase, business analysts model the business processes of the target STS with respect to security needs. As a first step, they derive business processes from the provided requirements document to create a BPMN 2.0 model. In a second step, they use SecBPMN2 to specify organizational security needs, again based on the requirements document.

**Phase 2.** Business process models deal with organizational aspects in a high level, abstracting from technical details.

Consequently, they are not sufficient for generating the system implementation directly. To this end, business processes and their included security requirements are now transformed to a secure system architectural model. We provide an automated model transformation from SecBPMN2 to UMLsec models. As a byproduct, this transformation creates a trace model of mappings to source and target elements, allowing us to keep processes model and system architecture aligned and bridge the gap between organizational requirements and technical ones. Sect. IV elaborates on the transformation implementation using the Henshin transformation language [26]. Since it is infeasible to foresee all desirable architectural details from the input business process, the generated output from this phase can be seen as a preliminary architecture model that needs to be revised by the system developers.

**Phase 3.** In this phase, system developers can refine the preliminary architecture model in two ways. First, missing details can be inserted, such as the associations names, attributes, and permissions of certain roles generated during Phase 2. Second, a UML element can be refined into multiple ones. For instance, classes with many contained operations can be split into several ones.




**Phase 4.** The architectural model enriched in phase 3 can be verified against the included security policies using CARiSMA [11]. While explaining the internal workings of the verification is outside the scope of this work, a short summary for the «*secure dependency*» case is found in Sec. II. The output *final report* contains the result of the check of each verified policy. If the architectural model does not satisfy all security policies, the architectural model needs corrections; the process then jumps back to step 3. If all security policies are correctly enforced, the security requirements engineers have evidence that the architecture is secure and meets the organizational security requirements specified in the business processes model. Therefore, one can guarantee that the system architecture is aligned with the business processes and can serve as basis for a secure implementation.

#### IV. SEC BPMN2 TO UML SEC TRANSFORMATION

The specification of system architectural model from business processes model is not straightforward in real-world STSs, which are large and complex. To address this challenge, we define an automated model transformation from

Procedural Model	Deployment Diagram	Class Diagram
Process	Node	-
Pool	Node	Class
SwimLane	Node	Class
Data Object	Artifact	Class
Task	-	Operation
Event	-	Operation
Message Flow	Communication Path	Association
Data Association	-	Association
Security Association	Dependency	Dependency

(a) SecBPMN2 elements to UML elements.

SecBPMN2 Security Annotations	UMLsec Security Policies	
	Deployment diagram	Class diagram
 Accountability	-	«abac» «abacRequire» {right}
 Confidentiality	«secure links» {adversary} «encrypted» «secrecy»	«abac» «abacRequire» {right} «secure dependency» «critical» {secrecy} «secrecy»
 Integrity	«secure links» {adversary} «encrypted» «integrity»	«abac» «abacRequire» {right} «secure dependency» «critical» {integrity} «integrity»

(b) SecBPMN2 security annotations to UMLsec security policies.

Fig. 4: Mapping Specification of SecBPMN2 to UMLsec structural diagram.

SecBPMN2 models to UMLsec structural diagrams (i.e., deployment and class diagrams), using the model transformation language *Henshin* and its associated toolset [26], [27]. Henshin is based on the Eclipse platform and the Eclipse Modeling Framework (EMF). It supports a graph-based specification of model transformation rules for arbitrary EMF-based meta-models. The rationale for using Henshin was its convenient application to our setting with respect to our goals, including the possibility to create a trace model during the transformation in order to manage traceability.

We focus on the part of SecBPMN2 that can be translated to an architectural model expressed using deployment and class diagrams. These diagrams enable the specification of security policies in different design views of the system.

**Mapping schema.** To specify the transformation rules systematically, we first define a mapping schema from SecBPMN2 to UMLsec elements. In Fig. 4a, the considered business model elements are linked to suitable types of deployment and class diagrams as follows:

- Since one cannot automatically identify from the SecBPMN2 model whether a set of processes is to be run on one node or a set of nodes, as a simple heuristic, we assume that each *Process* is running on a separate node, and therefore, is mapped to a *Node*. The node name is the name of the process, followed by "\_Subsystem".
- A role played by a participant (i.e., *Pool* or *Swimlane*) is mapped to a *Node*. The name of each node is the name of the corresponding participant followed by "\_Client".
- A *Message Flow* in SecBPMN2 is used to pass messages

between two processes. Since the processes are mapped to nodes in deployment diagram, a *Message Flow* is mapped to a *Communication Path* that carries the communications between the corresponding nodes.

- Each *Data Object*, identified by a name, is mapped to an *Artifact*<sup>1</sup>. The artifact name is the name of the data object followed by "\_Database".
- A role played by a participant (i.e., *Pool* or *Swimlane*) is mapped to a *Class*. The name of each class will be the name of the corresponding participant.
- In SecBPMN2, a security annotation is linked to a specific element by using a *Security Association*. In UMLsec, a dependency between the communicated nodes or classes is used to show the corresponding security requirements. Therefore, *Security Association* is mapped to a dependency in UMLsec diagrams.
- A role played by a participant (i.e., *Pool* or *Swimlane*) is mapped to a *Class*. The name of each class will be the name of the corresponding participant.
- Each *Data Object* is mapped to a *Class* of the same name. SecBPMN2 uses data objects to specify dataflow. To support a technical realization of this dataflow, the architectural class diagrams needs to provide appropriate classes, which are instantiated by objects in the running system. There can be multiple of these objects as repeated executions of the process require fresh objects.

<sup>1</sup>In earlier UMLsec versions, this information was expressed using components in deployment diagrams, which is not supported in UML2. We mapped the data objects to artifacts which are used to manifest system components.

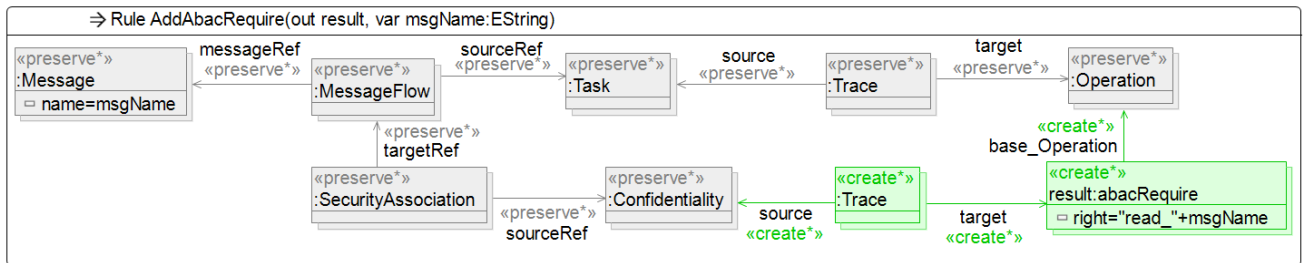


Fig. 5: Henshin rule for adding «abacRequire» UMLsec stereotype.



- Each *Task* or *Message Event* owned by a participant is mapped to an *Operation* in the corresponding class of the participant.
- A *Data Association* in SecBPMN2 is used to link a task or an event to a data object. The *Data Association* is mapped to an (1 : *n*) association between the class that represents the participant owning the task/event, and the class representing the data object.

The underlying background for our mapping schema benefits from related works aiming to relate BPMN with UML structural diagrams [28], [29], [30], [13]. However, we extended the mapping specifications proposed in these works for supporting the transformation to other UML elements that are needed for specifying security requirements, such as dependencies, communication paths, and artifacts. More details about the mapping specifications are provided as part of the transformation rule descriptions in [31].

In what follows, we explain our rationale for the mapping of SecBPMN2 security annotations, as shown in Fig. 4b.

*Confidentiality/Integrity (Data Object/Message Flow).* When attached to a data object or a message flow, *Confidentiality* and *Integrity* denote that the contents of data object or the message can be accessed or modified only by authorized users [3]. A UMLsec model aligned with these requirements needs to include three policies: First, a «*secure links*» policy in a deployment diagram, to check if confidentiality and integrity of the data is preserved during transmissions. As per Table I, encrypting the data on the communication links can guarantee the confidentiality and integrity of the data against default adversary, but it does not shield against insider attackers. Integrity and confidentiality of the corresponding data can be ensured via access control. To this end, second, the «*abac*» stereotype enforces the use of *Role-centric Attribute-Based Access Control*. Third, classes in a class model have dependencies via operation calls. To ensure that the dependencies between UMLsec classes respect the security requirements on the data communicated between them, the «*secure dependency*» can be used.

*Accountability/Integrity (Task):* The *Accountability* and *Integrity* security SecBPMN2 annotations can be applied to tasks. In this case, they express the need of monitoring a set of users when executing the task, and that the functionality of task should be protected from intentional corruption, respectively. These notations can be enforced by employing an access control mechanisms. Therefore, we transformed this security annotation to the UMLsec «*abac*» policy.

**Transformation rules.** Given the mapping schema and the existing EMF implementations of SecBPMN2 and UMLsec, we define a set of Henshin transformation rules. The rules are defined graphically and applied to the input model via an interpreter engine provided by Henshin. Each rule is specified in terms of a graphs pattern, where nodes and edges represent source and target models elements and their connections, respectively. Fig. 5 shows one of these rules, in which a *Confidentiality* annotation attached to a message flow is transformed to a «*abacRequire*» UMLsec stereotype. Each rule element has

an associated action, such as «*preserve*» or «*create*». Elements with a «*preserve*» action must be matched in the model to trigger the rule's application; elements with a «*create*» action describe modifications. *Trace* elements allow correspondences between source and target models elements to be maintained, as is key for establishing traceability of security requirements. Rule *AddAbacRequire* adds a «*abacRequire*» stereotype and its tagged value *right* to a given operation if a match for the whole *preserved* part is found. The trace element created together with the «*abacRequire*» is stored as part of a trace model, thus establishing traceability between the models. Our example models in Sect. II illustrate how this rule is applied.

The rules are divided into set of rule groups, being devoted to particular goals. For space reasons, an abbreviated account of all transformation rules are given in textual form. A full textual account is found in [31].

#### A. Deployment diagram transformation rules (DR):

Based on the mapping specification in Fig. 4, the transformation to deployment diagrams includes the transformation to deployment diagram elements, and to stereotypes related to the UMLsec «*secure links*» policy. For better readability the rules are divided into set of groups:

**Nodes and Communication paths (DR1).** Based on the mapping schema in Fig. 4a, this set is responsible for producing nodes, their deployed artifacts and communication paths from the respective SecBPMN2 elements.

**Security (DR2).** This set is responsible for transforming SecBPMN2 security annotations to «*secure links*» policy based on the mapping schema in Fig. 4b.

- **DR2.1** If a participant carries out a task or an event that manipulates a *Confidentiality*- or *Integrity*-annotated data object, the communication path between the corresponding client and subsystem node is stereotyped as «*encrypted*».
- **DR2.2** If a participant carries out a task manipulating a data object that is linked with a security association to a *Confidentiality* or *Integrity* annotation, the *Security Association* is transformed to a «*call, secrecy*»- or «*call, integrity*»-annotated *Dependency* from the *GUI* artifact of the participant to the corresponding artifact of the data object.
- **DR2.3** If two participants are communicating with each other via a *Confidentiality*- or *Integrity*-annotated *Message Flow*, the communication path between the corresponding subsystem nodes of the communicated participants is stereotyped as «*encrypted*». In addition, the communication path between the client and the subsystem node corresponding to each participant is stereotyped as «*encrypted*».
- **DR2.4** If two participants communicating with each other via a message flow that is linked with a security association to a *Confidentiality* or *Integrity* annotation, the *Security association* is transformed to a «*call, secrecy*»- or «*call, integrity*»-annotated *Dependency* between the application artifacts which are deployed on the corresponding subsystems nodes to the participants. In ad-

dition, a «*call, secrecy*»- or «*call, integrity*»-annotated dependencies are added between the *GUI* artifact and the application artifact corresponding to each participant.

- **DR2.5** A generated UML deployment model is annotated with «*secure links*» being tagged with {*adversary=default*}, if it has the at least two nodes communicated over an «*encrypted*»- or «*Internet*»-annotated paths with «*secrecy*»- or «*integrity*»-annotated dependency between their artifacts.

**B. Class diagram transformation rules (CR):** Based on the mapping schema, the transformation to class diagrams includes the transformation to class diagram elements, and to stereotypes related to UMLsec's «*abac*» and «*secure dependency*» policies. The rules are divided into a set of groups:

**Initialization (CR1).** This set is responsible for creating the UMLsec class diagram with some important core classes to aggregate information recurring across security requirements, thus improving readability.

**Classes (CR2).** Based on the mapping schema in Fig. 4a, this set is responsible for producing classes and their operations from the respective SecBPMN2 elements.

**Relationships (CR3).** This set is responsible for producing the relationships between the created classes, based on the mapping schema defined in Fig. 4a.

**Security (CR4).** This set is responsible for transforming SecBPMN2 security annotations to UMLsec «*abac*» and «*secure dependency*» security policies based on the mapping specification in Fig. 4b. In the following, we use the *TN*, *DN* and *MN* as abbreviations for *Task Name*, *Data Object Name* and *Message Name* respectively.

- **CR4.1** If a task is *Accountability*- or *Integrity*-annotated, the corresponding operation to the task is stereotyped with «*abacRequire*», being tagged {*right=access\_TN*} or {*right=modify\_TN*} in case of *Accountability* or *Integrity*, respectively.
- **CR4.2** If a participant carries out a task or an event manipulating a *Confidentiality*- or *Integrity*-annotated data object, (i) the classes for the participant and the data object are stereotyped as «*critical*» along with {*secrecy=TN*} or {*integrity=TN*}, (ii) the operation representing the task is stereotyped with «*abacRequire*», tagged {*right=read\_DN*} or {*right=modify\_DN*} in case of *Confidentiality* or *Integrity*, respectively, and (iii) a «*call, secrecy*»- or «*call, integrity*»- annotated *Dependency* between the classes for the data object and the participant is created.
- **CR4.3** If a participant carries out a task or an event being the source or the target for a *Confidentiality*- or *Integrity*-annotated message flow, the class for the participant is stereotyped with «*critical*» along with {*secrecy=TN*} or {*integrity=TN*}, (ii) the operation representing the task is stereotyped with «*abacRequire*», tagged {*right=read\_MN*} or {*right=modify\_MN*} with respect to the SecBPMN2 security annotation, and (iii) a *Dependency* stereotyped with «*call, secrecy*» or «*call,*

*integrity*» between the classes for the participants is created.

- **CR4.4** If the generated class model has at least one «*abacRequire*»-annotated operation, «*abac*» stereotype is attached to the generated class "RBAC", along with two tags {*role*} and {*right*}. The former specifies a set of roles, where roles are the names of all subclasses for "RBAC" class. The latter needs to be specified by the user after the transformation; it specifies the permissions associated to each role.
- **CR4.5** If the generated class model has at least one «*critical*»- annotated class that is being the source for a «*secrecy*»- or «*integrity*»-annotated dependency, «*secure dependency*» stereotype will be attached class model.

Following **CR4.3**, the example rule *AddAbacRequire* in Fig. 5 is used for producing the «*abacRequire*» stereotype with the {*right=read\_MN*} tag .

**Trace model.** As a prerequisite for managing traceability, our transformation rules create a trace model, consisting of traceability links between the input and output models. Fig. 6 represents the trace model generated as a result of applying the Henshin transformation rule in Fig. 5. The trace model links the SecBPMN2 and UMLsec models. Using the trace models, one can check whether a UMLsec security stereotype is in place for each security annotation specified in the SecBPMN2 model. For example, in Fig. 6 one can see that the *Confidentiality*-SecBPMN2 annotation is transformed to «*abacRequire*»-UMLsec stereotype.

## V. CASE STUDY

To study if the proposed approach satisfies the specified goals of *integrated management* and *traceability*, we applied it in a case study featuring the *System Wide Information Management* (SWIM, [32]) of the Federal Aviation Administration of the United States. SWIM is a technology program focusing on information sharing for *Air Traffic Management* (ATM) systems. ATM systems consist of a large number of autonomous and heterogeneous components that interact with each other to enable ATM operations: pilots, airports personnel, national airspace managers, weather forecast services, radars, etc. In such a complex information system, ensuring security is critical, for security leaks may result in severe

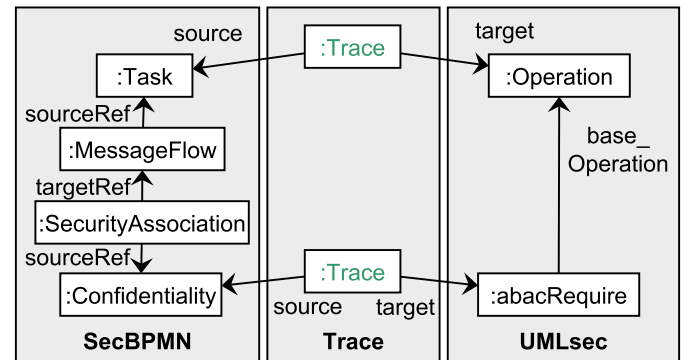


Fig. 6: Example of the generated trace models (excerpt).

consequences on safety and confidentiality. Below, we first describe and exemplify our application of the approach and then discuss if the goals are satisfied. The models created for the case study are provided online at <https://github.com/dstrueber/secbpmn2umlsec-casestudy>.

**SecBPMN2 models.** In the first step, we assumed the role of a business analyst in our process. Therefore, the task was to create SecBPMN2 models from the given requirements description. To this end, we analyzed the documentation provided with the case study focusing on three aspects: (1) *Flight plan negotiation*. Every time an airplane enters a new aerospace, a new flight plan is negotiated with the local authority. To model the set of procedures which regulate such negotiation we defined a SecBPMN2 model with 74 elements (data objects, tasks, events and data associations), 3 participants, and 22 security annotations (accountability, confidentiality and integrity). (2) *Landing*. Landing procedures are executed to negotiate the last part of the flight plan, which includes the approach to the airport and the waiting trajectory. We defined a SecBPMN2 model containing 84 elements, 4 participants, and 19 security annotations. (3) *External services*. The new ATM SWIM architecture permits to use external services for providing information such as the weather forecast. Services are selected based on a trust value which is updated every time a service is used and evaluated. For readability, we created two separate SecBPMN2 models for this aspect, containing 164 elements, 9 participants, and 30 security annotations in total.

**UMLsec models.** We applied our transformation to the three aspects of the ATM case study described above. The *Flight plan* and *Landing* aspects are each represented by a SecBPMN2 model, the *External services* aspect is represented by two models for readability. Based on the rules in Sect. IV, a UMLsec deployment and class models are automatically generated for each SecBPMN2 model.

To illustrate the outcomes, Fig. 2 shows the output model produced from the input model in Fig. 1. In the deployment diagram, following the specifications in **DR1** the processes have become subsystem nodes each with an application deployed artifact, while the participants are now client nodes, each with, GUI as a deployed artifact. For example, *Airplane*, a process with one participant, is transformed to two nodes; the first is the *Airplane\_Subsystem* node with a deployed artifact called *Airplane\_Process\_Application*, while the other is the *Airplane\_Client* node with *Airplane\_GUI* as a deployed artifact. Following the same specification in **DR1**, three communication paths are added and the *Flight plan* has become a *Flight plan\_Database* artifact deployed on the *Airplane\_Subsystem* node. Based on **DR2.3**, each of Communication Path1, Communication Path2 and Communication Path3 are stereotyped as «*encrypted*». Following the specification of **DR2.4**, a «*call,secrecy,integrity*»-annotated dependency is added between *Local authority\_Process\_Application* and *Airplane\_Process\_Application*. Accordingly, two «*call,secrecy,integrity*»-annotated dependen-

cies are added, one is between *Airplane\_GUI* and *Airplane\_Process\_Application*, while the other is between *Local authority\_GUI* and *Local authority\_Process\_Application*. As a final step for the transformation to the deployment diagram a «*secure links*» with {*adversary=default*} is added to the diagram based on **DR2.5**. Due to the fact that most UML modeling tools do not display tagged values of stereotypes in the diagram view, we show the tagged value as a note linked to the corresponding stereotype. The resulted diagram can be automatically verified against the «*secure links*» policy.

In the class diagram, following the specifications in **CR2**, each participant and data object has become a class. Classes are annotated with security annotations. For instance, as per **CR4.3**, both the *Confidentiality* and *Integrity* SecBPMN2 annotations linked to the *Flight plan* message flow in Fig. 1 are transformed to «*abac*» and «*secure dependency*» UMLsec security policies. First, the transformation to «*abac*» involved the attachment of «*abacRequire*» stereotype along with a tag {*right=read\_Flight plan, modify\_Flight plan*} to *Notify local authority()* operation. Again, we show the tagged values as notes connected to the corresponding stereotype. Second, following the same specification in **CR4.3**, the transformation to «*secure dependency*» stereotype involved the attachment of «*critical*» stereotypes to the corresponding classes of the participants along with {*secrecy=Notify local authority()*} and {*integrity=Notify local authority()*} as tags. Subsequently, the *Message Flow* is transformed to UML «*call*» *Dependency* stereotyped with «*secrecy*» and *integrity*. Finally, based on **CR4.1** the transformation of SecBPMN2 *Accountability* security annotation involved the attachment of «*abacRequire*» along with {*right=access\_check flight plan*} to the *check flight plane()* operation. Finally, based on **CR4.4-CR4.5** «*abac*» and «*secure dependency*» stereotypes are added to the generated *RBAC* class and to class diagram respectively.

**Refinement and validation.** Since the BPMN-based approaches do not contain the information required to generate complete UML models, our framework suggests a refinement phase where the system developers can split large classes and specify missing details, such as attributes and association names. Finally, once the refinement phase is completed, the system developers can verify the designed UMLsec class diagram against the specified security policies using CARiSMA. For instance, the deployment diagram in Fig. 2 allows the secrecy and integrity of the data communicated between *Airplane\_Client* and *Local authority\_client* nodes to be preserved against the default attacker pattern, see Table I. Moreover, as shown in Fig 2, using the «*abac*» policy, system developers can specify a list of rights for each role, and thereby, they can define the set of operations that can be accessed by a certain subject who plays a specific role in the system. For example, a given subject plays an airplane role has an access to the *Notify local authority()* operation. One can also observe that the *Airplane* class and «*call*» dependency provide the security level on the *Notify local authority()* operation that is required by *Local authority* class, and thus,



TABLE II: SecBPMN2 and UMLsec Models With the Numbers of: Participants, Data Objects, Message Flows and Accountability, Confidentiality, and Integrity annotations; Nodes, Classes, and Encrypted Paths, and Abac, AbacRequire, Critical, Secrecy, and Integrity annotations and Dependencies.

Model	SecBPMN2						UMLsec								
	Pa.	DO.	MF.	Ac.	Cf.	Ig.	No.	Cl.	EP.	Ab.	AR.	Cr.	ST.	IT.	Dp.
<i>Flight plan</i>	3	6	14	5	7	10	6	12	5	1	10	3	17	26	11
<i>Landing</i>	4	5	15	5	5	9	8	12	7	1	10	4	16	24	16
<i>External services 1</i>	4	4	5	3	2	4	8	11	7	1	6	4	6	11	10
<i>External services 2</i>	5	5	17	4	3	14	10	13	10	1	18	5	8	42	21

the security dependency is preserved. Per comparison between the source and target models in Figs. 1 and 2 respectively, one can guarantee that the security requirements are correctly traced and both models are aligned.

For space reasons, we summarize the overall transformation results in terms of metrics. Table II denotes the source and target models of each transformation. Each row represents one of the four model pairs *Flight plan*, *Landing*, *External services 1* and 2. For example, the *Flight plan* model contains 3 processes, each with one participant, and 5 *Accountability*, 7 *Confidentiality* and 10 *Integrity* security annotations.

In our input model, the *Confidentiality* and *Integrity* annotations are linked to 10 out of 14 message flows, where the sources of message flows were generally tasks, while the targets were message-receive events. To transform the security annotations, a «*encrypted*» stereotype is assigned to the communication paths between the corresponding communicated nodes. Based on **DR1** and **DR2.3**, our target model should include 6 nodes, and 5 «*encrypted*»-annotated communication paths as it is matched by the number in the table. In the class diagram, a «*abacRequire*» stereotype is assigned to the linked task (in case of *Accountability* **CR4.1**) and to the source and target task (in case of *Confidentiality* and *Integrity* **CR4.3**). In contrast to the tasks, the message-receive events are not human-controlled and, therefore, are not subject to access control in our transformation rules. 10 out of 23 tasks are a source of a *Confidentiality*- or *Integrity*-annotated message flow, and 5 of them are also *Accountability*-annotated tasks. Therefore, we should have in total 10 «*abacRequire*» stereotypes for the tasks and 1 «*abac*» stereotype attached to *RBAC* class, as it is matched by the numbers in the table. Again based on **CR4.3**, for each *Confidentiality* or *Integrity*, two «*critical*» stereotypes must be generated each with same tag type (i.e., *secrecy* or *integrity*) and values; the first will be assigned to the corresponding class of the sender while the second will be assigned to the recipient class. Since we have in total 3 classes for the participants, we expect the target model to have 3 «*critical*» stereotypes each with one *secrecy* and one *integrity* as tags. In total, the *secrecy* tags should have 17 values, while the *integrity* tags should have 26. Table II shows that this is the case.

In the input model, the annotated message flows can be grouped into four sender-receiver pairs: (1,2), (2,1), (2,3), and (3,2). Following the specifications of **CR4.3**, the number of «*secrecy*»- and «*integrity*»-annotated dependencies in the target class diagram depends on the number of these pairs: we should

have 4 dependencies in the class diagrams. Conversely, via **DR2.4**, the deployment diagram should include 7 «*secrecy*»- and «*integrity*»-annotated dependencies; 4 between the subsystems nodes and 3 between the clients and subsystem nodes. Therefore, our target model should have in total 11 «*secrecy*»- and «*integrity*»-annotated dependencies, as is the case in the table. Therefore, we can conclude that the SecBPMN2 security requirements are enforced in the UMLsec model.

**Discussion.** This case study demonstrates how our approach allows security requirements to be managed and traced throughout the development process.

*Integrated management* of security requirements across different phases is established via our automated transformation that integrates the involved languages. As business analysts, we focused on expressing the organizational requirements and security requirements. As system engineers, we could concentrate on designing and implementing the system architecture securely, without having to learn the specifics of business process modeling. From Table II, we can infer that the automated transformation saved us from re-implementing a large number of security annotations, which would have been daunting and error-prone.

*Traceability* is established via the trace models generated by our transformation rules. The trace models contain traceability links from source to target model elements. As one possible use for these links, an interested stakeholder could check whether a UMLsec security mechanism is in place for each security annotation specified in the SecBPMN2 model. Therefore, trace models are a promising means to increase trust in the produced models. Altogether, our case study highlights the potential benefits of integrating SecBPMN2 and UMLsec via our framework.

**Threats to validity and limitations.** Two main threats to external validity are that we emulated the involved expert stakeholders, rather than involving actual ones, and that we focus on a single case study. While a key benefit of our process is its reliance on notations familiar to the involved users, an empirical usefulness study is left to future work. We also aim to apply our process to a broader selection of cases. A threat to internal validity is the lack of a formal validation of the correctness of our transformation. Our transformation benefits from the capability to check the output models against UMLsec security policies, which, however, does not ensure that the intention of the business analyst is accurately reflected. Due to the large variability of possible SecBPMN2

models, our transformation could still be affected by errors. Finally, we focus on a subset of SecBPMN2 of those security annotations with an equivalent on the architectural level. We intend to increase coverage in the future. For instance, the *Non-Repudiation* SecBPMN2 annotation can be mapped to «provable» in UMLsec activity diagrams. Generating activity diagrams from BPMN models for this purpose would be a straightforward extension of our transformation.

## VI. RELATED WORK

**Model-based security analysis.** There has been a significant amount of work in the area of Model-based security analysis. An overview can be found in [33], which reviews existing approaches for the safety and security analysis of model-based object-oriented software designs, and identifies ways in which these approaches can be improved and made more rigorous. Some research addresses linking the model to the code level within model-based security through model-driven reverse engineering [34], [35]. Other work addresses the model-based use of security patterns [36], [37]. Further research makes use of aspect-oriented modeling for model-based security [38]. [39] proposes an approach for model-based security verification.

The specification of a UML system architecture model based on business processes represented by BPMN or UML activity models has been subject of previous works [28], [29], [30], [13]. However, except for the following ones, none of these works considered security aspects during the transformation.

**Automated transformation.** In [13], the authors used QVT [40] to specify a mapping from security-annotated UML activity models to UML class diagrams. The mapping is of a one-to-one kind, where each security requirement is transformed to a class or annotation with same name. Hence, as stated by the authors, the resulting security policies remain at a high level of abstraction, representing the view of business analysts. As result, security experts are needed for manually refining the security annotations to a technical security aspects. In contrast, our framework (i) supports the transformation to two UML structural diagram types in which security requirements at different design-level views of the system are captured, and (ii) does not require the involvement of security experts, since high-level security needs are transformed to verifiable security policies that encapsulate security knowledge. However, the presence of a security expert is still preferable. Conversely, the authors in [12] proposed a method to systematically develop UMLsec design models from security-annotated requirements models. The requirements models in their approach were UML models with *secure problem frame* annotations. While this approach makes the formal validation of UMLsec applicable, it is not tailored to business analysts, one of the main stakeholder types in socio-technical systems. In [18], the authors presented the transformation from misuse-cases [17] to mal-activity models [41]. Different from UMLsec, mal-activity models are not

verifiable against the specified activities and represents the view of business analysts.

**Manual transformation.** Other works have provided informal guidelines for addressing security while transitioning between stages in the development process. The authors in [42] proposed an approach to engineer the secure design of a system, using UMLsec [5], starting from organizational security requirements, using Secure Tropos [43]. For the same purpose, the authors in [44] proposed an approach to generate a UML class diagram represented by SecureUML [6] from organizational requirements represented by the KAOS [45] method. In [25], the authors connects UMLsec security policies with elicited requirements based on heuristics. This approach takes as input a set of requirements to predict a suitable UMLsec security policy for each security-related requirement. However, in these approaches, the system models must still be built manually and enriched with security stereotypes, a non-trivial and error-prone task.

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we present a framework for tracing high-level security requirements to verifiable technical security policies and, as a result, for bridging the conceptual gap between them. The main benefit is the management of security needs from the views of the involved expert stakeholders, in particular, business analysts and systems engineers, in an integrated manner. To this end, we integrate the two well-known model-based security approaches SecBPMN2 and UMLsec via model transformation. We illustrated the aforementioned benefits in a case study, in which our framework was suitable to render the early development stages of an air traffic management system less error-prone and more systematic. Our results are not restricted to any particular security extension of BPMN or UML, but can be applied to other ways of using BPMN and UML to address security requirements.

In the future, we aim to extend our approach to legacy situations, in which the UML design models are already given, rather than developed from scratch. Our mapping between SecBPMN2 and UMLsec security concepts can provide a foundation for addressing such legacy scenarios. Moreover, we aim to generalize our approach to the remaining SecBPMN2 annotation types by mapping them to various UMLsec diagrams. We also intend to extend the framework to incorporate the semantics of both languages, which would allow us to formally guarantee the correctness of our transformation. Finally, we also plan to perform a user study to study the usability of our framework.

## ACKNOWLEDGMENT

This research was partially supported by both the Deutsche Akademische Austauschdienst (DAAD) and the research project Visual Privacy Management in User Centric Open Environment (supported by the EU's Horizon 2020 program, Proposal number:65364)

## REFERENCES

- [1] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige, "Large-scale complex IT systems," *Commun. ACM*, vol. 55, no. 7, pp. 71–77, 2012.
- [2] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel, "SecureBPMN: Modeling and enforcing access control requirements in business processes," in *ACM Symposium on Access Control Models and Technologies*. ACM, 2012, pp. 123–126.
- [3] M. Salnitri, F. Dalpiaz, and P. Giorgini, "Modeling and verifying security policies in business processes," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2014, pp. 200–214.
- [4] M. Salnitri, "Secure Business Process Engineering: a socio-technical approach," Ph.D. dissertation, University of Trento, 2016.
- [5] J. Jürjens, *Secure systems development with UML*. Springer Science & Business Media, 2005.
- [6] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *International Conference on the Unified Modeling Language*. Springer, 2002, pp. 426–441.
- [7] A. Kennedy, K. Carter, W. Frank, and D. Architects, "MDA Guide Version 1.0," Tech. Rep., 2003.
- [8] N. MacDonald, "Model-driven security: Enabling a real-time, adaptive security infrastructure," <https://www.gartner.com/doc/525109/modeldriven-security-enabling-realtime-adaptive>, Gartner, Tech. Rep., 2007.
- [9] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "Analysis-level classes from secure business processes through model transformations," in *International Conference on Trust, Privacy and Security in Digital Business*. Springer, 2007, pp. 104–114.
- [10] "BPMN 2.0," <http://www.omg.org/spec/BPMN/2.0/>.
- [11] "CARiSMA," <https://rgse.uni-koblenz.de/carisma/>.
- [12] D. Hatebur, M. Heisel, J. Jürjens, and H. Schmidt, "Systematic development of UMLsec design models based on security requirements," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2011, pp. 232–246.
- [13] A. Rodríguez, E. Fernández-Medina, J. Trujillo, and M. Piattini, "Secure business process model specification through a UML 2.0 activity diagram profile," *Decision Support Systems*, vol. 51, no. 3, pp. 446–465, 2011.
- [14] M. Menzel, I. Thomas, and C. Meinel, "Security requirements specification in service-oriented business process management," in *International Conference on Availability, Reliability and Security*. IEEE, 2009, pp. 41–48.
- [15] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A BPMN extension for the modeling of security requirements in business processes," *IEICE transactions on information and systems*, vol. 90, no. 4, pp. 745–752, 2007.
- [16] "UML 2.0," <http://www.omg.org/spec/UML/2.0/>.
- [17] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements engineering*, vol. 10, no. 1, pp. 34–44, 2005.
- [18] M. El-Attar, "From misuse cases to mal-activity diagrams: bridging the gap between functional security analysis and design," *Software & Systems Modeling*, vol. 13, no. 1, pp. 173–190, 2014.
- [19] M. Alam, R. Breu, and M. Hafner, "Model-driven security engineering for trust management in SECTET," *JSW*, vol. 2, no. 1, pp. 47–59, 2007.
- [20] X. Jin, R. Sandhu, and R. Krishnan, "Rabac: role-centric attribute-based access control," *Computer Network Security*, pp. 84–96, 2012.
- [21] J. Jürjens, "Modelling audit security for smart-card payment schemes with UMLsec," in *16th International Conference on Information Security (IFIPSEC'01)*, IFIP. Kluwer, 2001, pp. 93–108.
- [22] J. Jürjens and G. Wimmel, "Formally testing fail-safety of electronic purse protocols," in *16th International Conference on Automated Software Engineering (ASE 2001)*. IEEE, 2001, pp. 408–411.
- [23] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens, "Enhancing security requirements engineering by organisational learning," *Requirements Engineering Journal (REJ)*, vol. 17, no. 1, pp. 35–56, 2012.
- [24] A. S. Ahmadian, D. Strüber, V. Riediger, and J. Jürjens, "Model-based privacy analysis in industrial ecosystems," in *European Conference on Modelling Foundations and Applications 2017*, 2017, pp. 215–231.
- [25] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider, "Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec," *Requirements Engineering*, vol. 15, no. 1, pp. 63–93, 2010.
- [26] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place EMF model transformations," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 121–135.
- [27] D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehler, M. Ohrndorf, and M. Tichy, "Henshin: A usability-focused framework for emf model transformation development," in *International Conference on Graph Transformations*, 2017, pp. 125–141.
- [28] N. Debnath, C. A. Martinez, F. Zorzan, D. Riesco, and G. Montejano, "Transformation of business process models BPMN 2.0 into components of the Java business platform," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*. IEEE, 2012, pp. 1035–1040.
- [29] A. Kriouile, N. Addamsiri, T. Gadi, and Y. Balouki, "Getting the static model of PIM from the CIM," in *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in*. IEEE, 2014, pp. 168–173.
- [30] Y. Rhazali, Y. Hadi, and A. Mouloudi, "A New Methodology CIM to PIM Transformation Resulting from an Analytical Survey," in *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, 2016, pp. 266–273.
- [31] Q. Ramadan, M. Salnitri, D. Strüber, J. Jürjens, and P. Giorgini, "From Secure Business Process Modeling to Design-Level Security Verification (long version)," <https://www.dropbox.com/s/0cmbgod3arze3vo/long-version.pdf?dl=0>.
- [32] F. A. Administration, "SWIM ATM case study, last visited November 2016," <http://www.eurocontrol.int/swim>.
- [33] K. Lano, D. Clark, and K. Androutsopoulos, "Safety and security analysis of object-oriented models," in *Computer Safety, Reliability and Security, 21st International Conference, SAFECOMP 2002, Catania, Italy, September 10-13, 2002, Proceedings*, 2002, pp. 82–93.
- [34] S. M. Perez, J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot, "Model-driven extraction and analysis of network security policies," in *MODELS 2013*. Springer, 2013, pp. 52–68.
- [35] S. Martínez, V. Cosentino, and J. Cabot, "Model-based analysis of Java EE web security configurations," in *Proceedings of the 8th International Workshop on Modeling in Software Engineering*. ACM, 2016, pp. 55–61.
- [36] B. Katt, M. Gander, R. Breu, and M. Felderer, "Enhancing model driven security through pattern refinement techniques," in *FMCO 2011*, 2011, pp. 169–183.
- [37] 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 *MoDELS 2015*, 2015, pp. 246–255.
- [38] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. H. Houmb, "An aspect-oriented methodology for designing secure applications," *Information & Software Technology*, vol. 51, no. 5, pp. 846–864, 2009.
- [39] C. L. Heitmeyer, M. Archer, E. I. Leonard, and J. McLean, "Applying formal methods to a certifiably secure software system," *IEEE Trans. Software Eng.*, vol. 34, no. 1, pp. 82–98, 2008.
- [40] "QVT 1.3," <http://www.omg.org/spec/QVT/1.3/PDF>.
- [41] G. Sindre, "Mal-activity diagrams for capturing attacks on business processes," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2007, pp. 355–366.
- [42] H. Mouratidis and J. Jürjens, "From goal-driven security requirements engineering to secure design," *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 813–840, 2010.
- [43] H. Mouratidis, "A security oriented approach in the development of multiagent systems: applied to the management of the health and social care needs of older people in England," Ph.D. dissertation, University of Sheffield, 2004.
- [44] Y. Ledru, J.-L. Richier, A. Idani, and M.-A. Labiadh, "From KAOS to RBAC: A case study in designing access control rules from a requirements analysis," in *Conference on Network and Information Systems Security*. IEEE, 2011, pp. 1–8.
- [45] A. Van Lamsweerde, *Requirements engineering: from system goals to UML models to software specifications*. Wiley Publishing, 2009.