Maintaining Security Consistency During System Development with Security-Oriented Model Federation

Chahrazed Boudjemila, Fabien Dagnat, Salvador Martínez

Maintaining Security Consistency During System Development with Security-Oriented Model Federation

Chahrazed Boudjemila
chahrazed.boudjemila@imt-atlantique.fr
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France

Fabien Dagnat
fabien.dagnat@imt-atlantique.fr
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France

Salvador Martínez
salvador.martinez@imt-atlantique.fr
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France

ABSTRACT

Multi-modeling is an approach within the MDE realm that promotes the development of complex systems by decomposing them in sets of heterogeneous models. These models are defined using different modeling languages and constructed using diverse tools. They represent different but often interdependent *views*. However, the models of a system are far from being static. They change to accommodate new requirements, functionality improvements, bug fixes, and other evolution events. These changes represent a challenge w.r.t. consistency. This is especially true in security-critical scenarios. Indeed, security information is often integrated within the systems models so that security requirements are met following what is called "security-by-design". In such scenarios, the security concern of the systems models must remain consistent across changes so that security properties continue to hold.

In order to tackle this problem, we propose a methodology to enhance the (multi)model-based design phase of a system development process. It comprises the creation of a security federation in which security dependencies between the different models are reified and equipped with *security rules* expressing security consistency requirements. Then, whenever a model is changed, the security rules are evaluated to monitor the consistency of security across the system models. We evaluate the capabilities of this methodology by a prototype implementation and its application to different use cases.

CCS CONCEPTS

• **Software and its engineering** → *Software development process management*; **System modeling languages**; • **Security and privacy** → **Software security engineering**.

KEYWORDS

Model-driven engineering, model federation, security by design, model evolution.

ICSSP '24, September 4–6, 2024, München, Germany

ACM ISBN 979-8-4007-0991-3/24/09.
https://doi.org/10.1145/3666015.3666016

ACM Reference Format:
Chahrazed Boudjemila, Fabien Dagnat, and Salvador Martínez. 2024. Maintaining Security Consistency During System Development with Security-Oriented Model Federation. In *International Conference on Software and Systems Processes (ICSSP '24), September 4–6, 2024, München, Germany*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3666015.3666016

1 INTRODUCTION

In the model-driven engineering (MDE) paradigm, the design phase of the development process of large and complex software and system projects often requires using multiple modeling languages (and/or tools). Each modeling language covers a different view of the application and is suitable for a corresponding set of domain experts. This approach is generally called *multi-modeling* [1] (also *intermodeling* or *view modeling*). Models participating in a multi-model system are, however, not isolated as (often implicit) semantically rich relationships may exist between elements of different models (*e.g.*, a relation may mean equivalence, dependency, specialization, etc.). As a consequence, maintaining consistency [10] is a critical and challenging issue when models evolve. A change in one model may require modifications of dependent models. This, in turn, requires the ability to propagate changes among models and evaluate these changes w.r.t. the semantics of the relationships.

When multi-modeling occurs in a security-critical scenario, the consistency challenge becomes even more crucial. Indeed, since security plays a vital role in many nowadays systems, several approaches aim to integrate security aspects into modeling and modeling languages (*e.g.*, UMLsec [4] and SecBPMN [22] among others) and adopt a *security by design* perspective to deal with security since the early phases of the development process. In this scenario, inconsistencies regarding security requirements and properties may lead to the inadvertent introduction of security flaws that make the modeled systems vulnerable to security threats.

A solution to this challenge requires at least the means to: 1) make explicit and exploitable the relationship between the models participating in a multi-model, notably w.r.t. security; 2) evaluate how a change propagates through these relationships and how that affects security. We propose to leverage the model federation to tackle this problem. The model federation paradigm promotes the reification of dependencies between heterogeneous models while keeping them in their original technological space [9], which partially fulfills the first requirement. Additionally, dependencies in model federation have customizable semantics and may carry behaviors which help to fulfill the second requirement and complete the first one.

Concretely, we propose a methodology for the creation, operation and management of what we call, *security federations*. Our methodology integrates within the (model-based) design phase of

the system's development process by adding new tasks and artifacts that help maintain security. It involves first the creation of a *security-oriented model federation* in which the security dependencies between the models are reified and equipped with *security rules*. These rules encode the conditions required for dependent elements in different models to remain consistent w.r.t. security properties. Second, and when the models change due to an evolution event, *security federation* is used to determine whether the changes impact the security of *the system under study*. To do this, the security rules attached to the reified dependencies that are related to the change are (re) evaluated. The results of the evaluation together with the reified dependencies inform the *security federation* user of the security impact of the change (if any) and gives information regarding the elements involved in the inconsistency so that consistency can be restored.

In this paper we extend preliminary work [3] in which we outline the first steps of the methodology. Specifically, compared to the previous work, this paper provides: 1) a detailed and formalized description of our methodology, offering comprehensive steps; 2) an evaluation introducing numerous new security rules covering various security properties of two use cases and new evolution scenarios; 3) a complete prototype implementation; 4) a critical discussion of our methodology and its evaluation.

The rest of the paper is organized as follows. Section 2 discuss the related work. A detailed description of our methodology's steps and rationale follows in Section 3. Section 4 presents our validation and use cases, including a discussion of limitations. Finally, we conclude the paper in Section 5 by summarizing our contributions and outlining future research directions.

2 BACKGROUND & RELATED WORK

This section is divided into three subsections. The first two subsections introduce the main concepts behind our work, namely model federation and security by design. The final subsection presents various approaches developed in the context of multi-model consistency and security.

2.1 Model Federation

As systems undergo frequent updates, managing the models in a multi-model system is a critical challenge. In recent years, various approaches have been developed in this field [15], including integration (e.g., through a pivot model and a set of model transformation), unification which relies on a general meta-model capable of having all the required models as instances, and model federation which promotes the reification of dependencies[9], enabling the integration and the management of diverse models that participate in a given federation. The model federation paradigm has two notable features. First, the models federated can independently evolve in their original technical spaces while remaining connected to the overall federation through adapters. Second, model federations are not merely static; on the contrary, the reified dependencies and the federation itself may be equipped with (automatic) behaviors. These behaviors may be used to give an intention to the federation (e.g., maintaining synchronization, consistency or security).

Due to the aforementioned features of the model federation paradigm, we chose it as infrastructure for our approach to maintain security consistency in multi-modeling scenarios. To achieve this objective, it is also essential to integrate security requirements into the design phase to identify and address security inconsistencies. Therefore, our research also builds on foundational principles that integrate the security by design paradigm.

2.2 Security-by-Design

Security by design paradigm is particularly relevant, as it emphasizes considering security requirements from the very beginning of the design phase. Its main objective is to identify and address potential security vulnerabilities and risks [18] of the system under development as early as possible.

In a MDE scenario, several different techniques may be used for the integration of the security-by-design paradigm. These include the creation of new, security-specific models (*e.g.*, attack and threat models) and the extension of existing modeling languages to include security concerns. Examples of the latter are UMLsec [12] and SecBPMN [22] where security is added to the widely used UML and BPMN modeling languages. We use both UMLsec and SecBPMN in the use cases described in Section 4. UMLSec is an UML profile (and an accompanying analysis tool) containing security-related stereotypes, tagged values, and application constraints. Similarly, SecBPMN introduces security to BPMN through a number of security annotations formalized with predicates.

2.3 Multi-model consistency

In the last two decades, many different approaches have been developed to deal with multi-modeling consistency [24, 25]. Due to space limitations, we only mention a few of the existing contributions ,prioritizing those more similar to model federation.

In [14] the authors present an approach to detect and repair inconsistencies across heterogeneous models. It is based on a set of general relationship between models and a classification of possible violations to those relationships. It is based on the epsilon ecosystem. Similarly, but with a stronger focus on the automation of the matching and change detection processes in [6] and [7] the authors describe a multi-model evolution approach and tool. In [16] the authors present an approach for managing design model inconsistencies caused by changes. This approach analyzes the changes made to models and automatically generates potential repairs. These repairs are then analyzed to identify any newly introduced inconsistencies. Their repairs can be integrated into our process when security inconsistencies are detected. More in the scope of model federation, in [13], the authors define a concept meta-model to establish inter-model relations called Commonalities. From a more theoretical point of view, in [23] the authors present *Comprehensive systems*, a structure based on category theory in which multi-model systems can be described, demonstrating that existing consistency verification and repair approaches may be used over it. None of these contributions deals with security directly, although they could arguably be adapted to deal with it, *e.g.*, by implementing our methodology. Besides, while they deal with semantic heterogeneity, they do not directly cross technological spaces. We choose Openflexo as our model federation infrastructure due to its flexibility and maturity and the availability of ready-to-use technological adaptors.

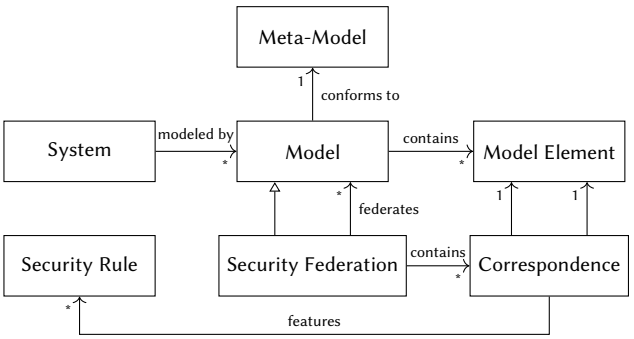

Figure 1: Methodology concepts meta-model

Focused on security and/or evolution, in [20] the author proposes an approach on top of UMLsec to study the evolution of system artifacts (conceptual level to program level) by using inter-artifact tracing and graph transformation techniques to synchronize models. Similarly, and also based on UMLSec, in [5, 19] the authors present an approach to co-evolve system models w.r.t. changes in security knowledge and requirements. Set of rules are used to both, detect changes and semi-automatically restore system security. Furthermore, in [21] the authors propose to bridge business process security design and technical security measures by using a set of transformation rules. We think that (parts of) their approach may be integrated into ours to enhance it. As an example, their rules may be used as behaviors in our links. In [8] the authors aim to integrate security across the entire engineering process so that security requirements can be traced to countermeasures (*e.g.*, in deployment). This approach and ours may be combined so that ours gets more security information and artifacts and their gets support for evolution. With the aim to automate security processes in software development, in [26] the authors developed an automated test mechanism for microservices which combines multiple open-source scanning tools to identify security vulnerabilities and suggest improvements. Finally, in [11] the authors propose an ontology based approach to automate the process of orchestrating various security systems tools. While their approach, as ours, also describes security relevant correspondences, they are used for a different purpose and in a much later phase.

In the following section, we present our work, which is based on the key concepts of model federation and security by design.

3 METHODOLOGY

This section describes our proposed methodology. It is based on the model federation paradigm and its objective is to provide the means to maintain the consistency of a multi-model *system under study* w.r.t. its security. In a first phase, our methodology prescribes the creation of a so-called *security federation*. This is achieved by establishing security relevant correspondences and specifying/implementing security rules exploiting them. Then, in a second phase, our methodology describes how to use the *security federation* to maintain security consistency when models change.

Our methodology places itself as a complement to the design phase of software and system's development process. It does so by involving new actors, prescribing new tasks and producing/ consuming new artifacts. When using our methodology, the user must first choose its development process and extends it with the new tasks described in this section.

To motivate the purpose of our methodology, we present a simple example of a medical system. This system is designed using two models: a data model and a BPMN model. The data model captures the essential data, their structure, and the relationships, while the BPMN model describes the process of obtaining a medical record. Some elements of the BPMN model correspond to some elements in the data model. For example, there is a Class element Prescription in the data model corresponding to a DataObject PrescriptionObject in the BPMN model. The two models can evolve independently. For instance, a designer may add a Privacy security annotation to the Prescription in the data model to specify that it represents personal data. If the BPMN model does not change, the privacy of the prescription may be violated since the corresponding PrescriptionObject do not require any specific security action. To maintain security consistency, the BPMN model should also specify that the PrescriptionObject is private.

This section provides a detailed explanation of our methodology for dealing with the aforementioned security consistency problem. It starts by defining the concepts it uses. Then, we present the creation of security-oriented model federations. Lastly, we introduce how these security federations are used. Section 4 illustrates this methodology on use cases. The proposed methodology is not dependent on specific models or model-federation approaches.

3.1 Methodology Meta-Model

The upper part of the conceptual model in Figure 1 describes the usual elements of systems design. A System is modeled by a set of Models conforming to various Meta-Models. A Model contains Model Elements. Reusing the example described above, Prescription and PrescriptionObject are both modeling elements of respectively the data model (conforming to the UML class diagram metamodel) and the process model (conforming to BPMN).

The bottom part defines the concepts of our methodology. Its central element is the Security Federation. It is a model federation, as such, it is a model and federates a set of models. It contains a set of Correspondences, a correspondence relating two model elements (generally of two different models) and featuring a set of Security Rules. Each a security rule specifies a security constraint that must be maintained between the two elements of the correspondence. In the above example, the security federation should at least contain a correspondence between the Prescription class and the PrescriptionObject data object. It should be equipped with a security rule ensuring that whenever the Prescription class features a Secrecy security annotation, the corresponding data object (here PrescriptionObject) also features the Integrity or Confidentiality security annotation.

3.2 Create a Security-Oriented Model federation

The creation of a security federation follows the process described in Figure 2. This process involves three actors. The designers are in charge of developing the models of the *system under study* to satisfy both its functional and non-functional requirements. The

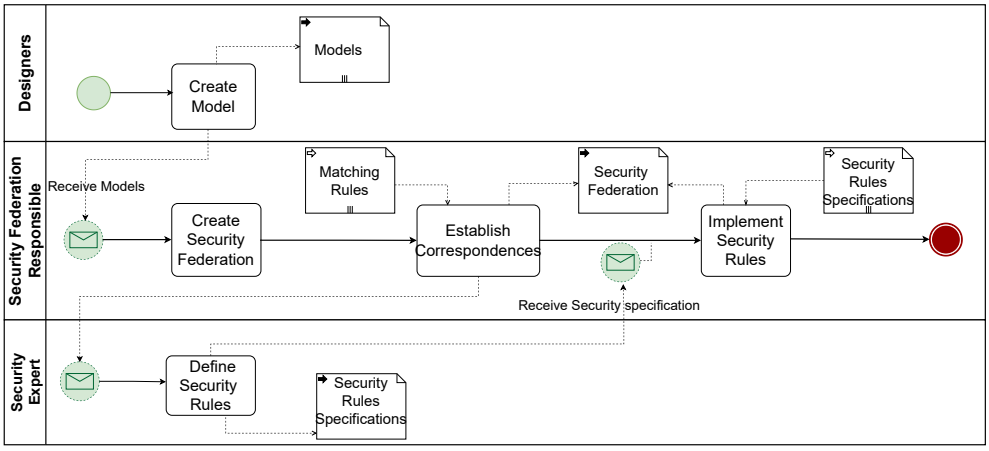

Figure 2: Process for the creation of a security federation

security federation responsible is in charge of developing and managing the security federation during the system development process. This includes the selection of the federation tool, the establishment of correspondences between elements of various models and the implementation of the security rules using the chosen federation tool. Finally, the security expert specifies inter-model security constraints. These three actors collaborate as described below.

First, the designers construct the various system models (task Create Models). Each designer describes his/her model using a language according to his/her domain expertise.

Then the security federation responsible can federate these models. Notice that this actor must be familiar with the model federation paradigm and the chosen tools for the federation. He/she creates the security federation using the chosen tools and starts the collection of the correspondences (task Establish correspondences). Our methodology focuses on establishing security-related correspondences, requiring a detailed examination of additional security information at the model level. To elicit the correspondences this actor may follow one of the numerous approaches that aim to identify correspondences between heterogeneous models. For example, one can follow [6, 23] and create a model or a meta-model of correspondences or [2, 21] and define a set of transformation rules that explicitly identify the correspondences. In both approaches, the correspondences are identified by using a set of relationships between the elements of the models' meta-model. These relationships named *matching rules* in the rest of this article may guide in establishing correspondences. An example, of such a matching rule is that a `Class` in a data model which contains security annotation (expressed as a UML class diagram) may correspond to a `DataObject` in the BPMN model. Applied to our medical system example, this rule helps in identifying the correspondence between the Prescription class and the PrescriptionObject dataObject. In this article, we assume that the matching rules exist and provide a set of correspondence candidates.

Once the responsible security federation finishes reifying all correspondences identified into the security federation, the security expert can start specifying the security rules. These security rules are defined according to the security requirements of the system and the security information added to the models (following security by design). They embody the inter-model security constraints and must be executed to evaluate security. Our security rules serve as indicators for users, notifying them of any security violation caused by security inconsistencies within system models when changes occur. They can be implemented with Java, OCL or any other suitable language. Moreover, it is crucial to note that while these rules are only evaluated in the context of this paper, other uses (*e.g.*, satisfiability analysis) may constrain or be constrained by the chosen language.

Each security rule is related to a correspondence and therefore to at least a pair of meta-models. Thus, it is highly dependent on the different chosen modeling languages and their security features. As stated at the end of the previous subsection, the correspondence between the Prescription class and the PrescriptionObject data object features a security rule ensuring that they contain (semantically) matching annotations.

Finally, the security federation administrator implements the security rule specification received from the security expert (task Implementing security rules). For this task, he/she must use a predicate language supported by the federation tool and attach them to the right correspondence.

3.3 Using the Security-Oriented Model federation

We have presented above the process to build a security federation. The aim of such a federation is to assist in the identification of security inconsistencies when any of the federated models involved evolve. Figure 3 presents the process of using the security federation to achieve this.

As already mentioned, models change over time to satisfy technical or business needs such as requirements changes, the introduction of new technologies, deployment changes, discovering new vulnerabilities, new regulations, etc. We consider that the process of using the security federation starts when a designer changes one of the system models. Such evolution may involve adding, removing, or modifying elements of a model. When a change occurs, the security federation responsible analyzes it. The evolution may

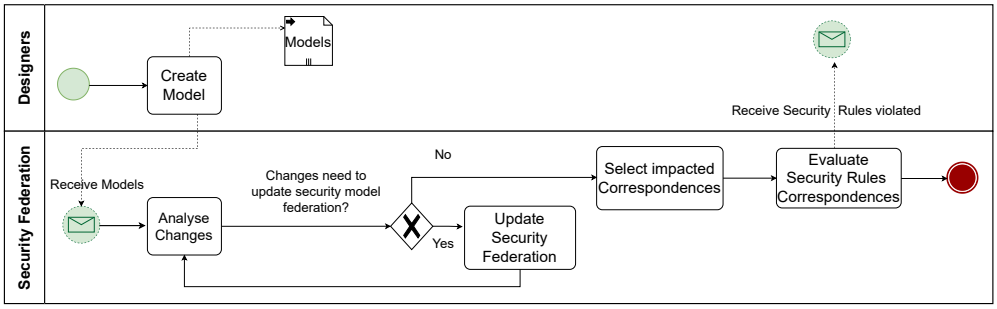

Figure 3: Process for using a security federation

need an update of the security federation, for example, a new correspondence must be defined. After a possible extension of the security federation, the security rules of the correspondences impacted by the change must be evaluated. This evaluation can be automatically initiated. Each re-evaluated security rules that is violated is sent to the designers together with the impacted elements. In the example cited in the beginning, we may add a security rule to ensure that the related elements incorporate semantically the same security annotations. When an annotation is added to Prescription, an evaluation of the security rule added to the correspondence between Prescription and the PrescriptionObject data object occurs.

4 EVALUATION

We dedicate this section to the evaluation of our methodology. This evaluation consists in its application to different use cases. For brevity, we detail here two of them, namely, *iTrust* and *Holiday Booking*. The rest are available on the website of the project[1]. Our methodology is implemented as a prototype on top of Openflexo[2], a mature state-of-the-art model federation framework. In order to ease the discussion in the following, we briefly introduce the main concepts behind Openflexo. The use cases are detailed in Sections 4.1 and 4.2 followed by a discussion in Section 4.3.

Openflexo provides us with the infrastructure required to connect multiple models while keeping each model in its original technological space. This infrastructure includes a Federation Model Language (FML) engine and an integrated model design environment. The main construct in Openflexo is the *Virtual Model* which corresponds to a model in more traditional modeling frameworks. Virtual models contain *FlexoConcept* elements (which can be seen as classes), which in turn carry *FlexoProperty* elements (*i.e.*, structural features). Flexo properties may use a *ModelSlot* to access information in external models (*e.g.*, the federated models) through so-called *technology adapters*. (Openflexo provides off-the-shelf technology adapters for BPMN, Excel, XML, EMF and OWL among others.) Figure 4 contains a screen capture of the FML editor illustrating the various elements explained above.

[1] https://github.com/labsticc-p4s/SecurityFederationModelsValidation
[2] https://openflexo.org

4.1 iTrust

The iTrust is an open-source medical system [17] developed in 2005 at the University of North Carolina with the aim of providing software engineering students with a complex real-world system for experimentation. In this sense, it has already been used in many different research works as a realistic use case. Regarding security, iTrust has requirements related to privacy, confidentiality, and integrity.

In the following we illustrate the application of our methodology to (parts of) iTrust. We rely on available models in literature [5] and on the iTrust public documentation. Among all the possible aspects and models of the systems, our use case relies on four models presented in Figure 5: 1) a BMPN model containing descriptions of data access and manipulation processes; 2) a UML class model defining the static structure of iTrust; 3) a UML deployment model, describing how iTrust is installed in a given environment, and 4) an access control model which describes roles and permissions on the system. Security concerns are represented in the BMPN and UML models by reusing the annotations of SecBPMN and UMLsec.

Once the models are created and selected to be part of the *security federation*, the next step of our methodology is the creation of correspondences between elements in different models. As we have mentioned, security-relevant correspondences are identified through the analysis of the security information integrated within elements of models (*e.g.*, security annotations) and the use of matching rules. Such a matching rule specifies elements in different models that may be in correspondence. For example, there is a matching rule that specifies that a `Class` in a data model of a system (expressed as a UML class diagram) may correspond to a `DataObject` in a BPMN model of the same system. Indeed, as they both are models of the same system, their elements may represent the same "thing". This matching rule may be applied to a model UMLsec and a model secBPMN. If a `Class` of the data model (in UMLsec) is annotated `Critical` and a `DataObject` (in secBPMN) is annotated by `ConfidentialityDO` or `PrivacyDo`, we can identify a security-relevant correspondence between the `DataObject` and the `Class`.

Referring to the commonalities identified between BPMN and data models as discussed in [23] and the set of transformation rules defined in [2, 21]. We show in the following, some of the matching rules that are relevant to the context of the iTrust case study.

- *MatchingRule-1*: A class in the UML Class model may correspond to a DataObject in the BPMN model.
- *MatchingRule-2*: A User in the UML Class model may correspond to a Role in the Access Control model.
- *MatchingRule-3*: A Role in the Access Control model may correspond to a Pool in the BPMN model.

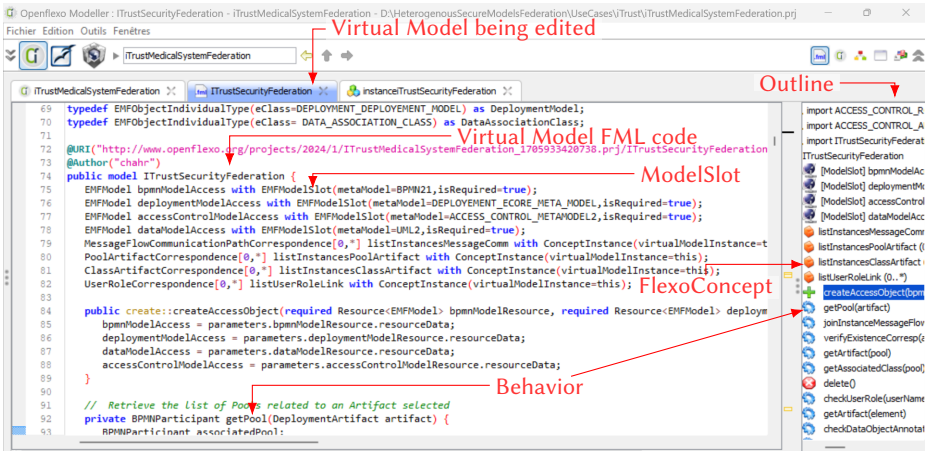

Figure 4: Openflexo Framework

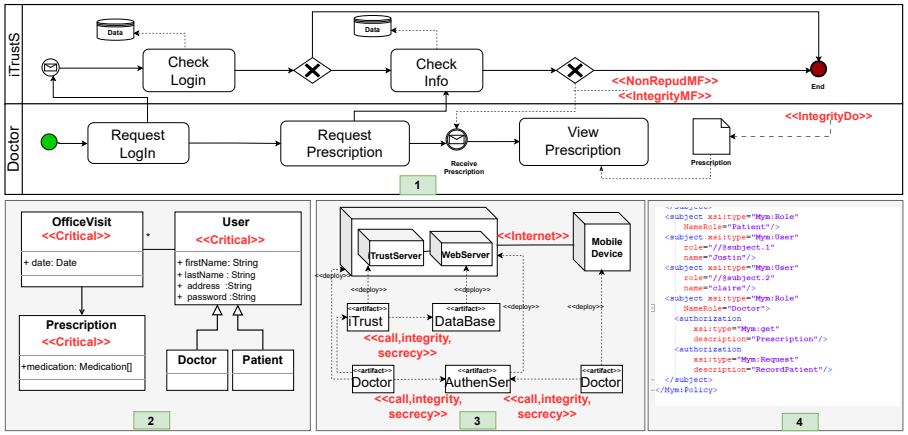

Figure 5: Excerpt of iTrust Medical System models

- *MatchingRule-4*: A permission in the Access Control model may correspond to an operation in the UML Class model.
- *MatchingRule-5*: The flow between two Pools in the BPMN model may correspond to the Association between Artifacts in the Deployment model.
- *MatchingRule-6*: A communication Link in the BPMN model may correspond to a communication Path in the Deployment model.

Figure 6 shows the result of the previously described task of finding the correspondences on the case study. Concretely, it shows three correspondences between elements of the BPMN and Deployment models. The two pools (Doctor and iTrust) are linked to their corresponding artifacts in the deployment model. The exchange between the doctor and the iTrust software of prescription is linked to the internet link between the server and the mobile device. Indeed, the Doctor artifact may be deployed on the mobile device while iTrust is deployed on the server.

To effectively create the *security federation* in the Openflexo framework, we proceed as follows. First, we create a virtual model called ITrustSecurityFederation to serve as the main interaction model, that is, the model that captures the dependencies between model elements of the different federated models. Practically, this is achieved with the creation of *Flexo concepts* and the corresponding *Flexo properties* using *Model slots*. For example, to link the pool element of the BPMN model with the artifact element of the deployment model, we have created a concept called PoolArtifactCorrespondence with two model properties using model slots relying on the corresponding technology adapters (BPMN and UML).

For the correspondences to be useful for monitoring security consistency, we need to add security rules encoding this security consistency requirement within the identified correspondences. For the iTrust system, we have defined 17 security rules based on its security requirements and using the four available models. They aim to ensure consistency of various security aspects concerning for example integrity, authorization, and non-repudiation throughout the iTrust system models. We did not conduct a detailed analysis to confirm the completeness of our security rules. Below there are

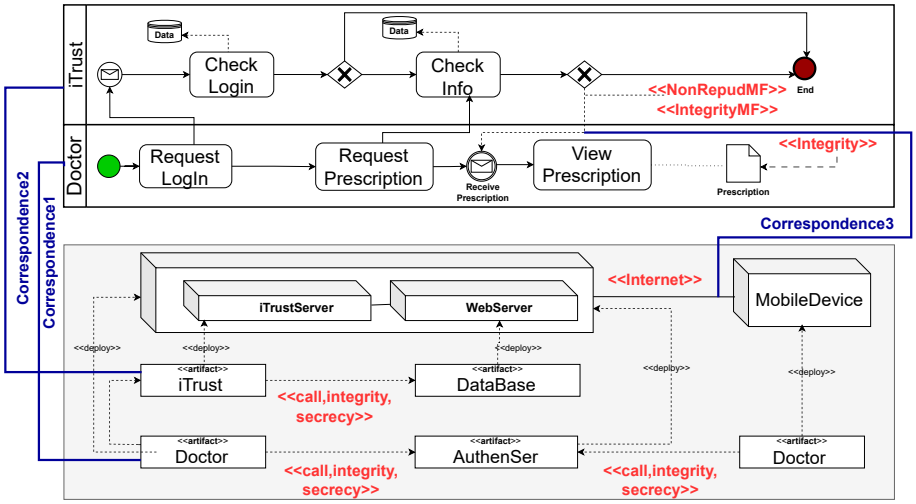

Figure 6: Excerpt of iTrust Medical System and the correspondences between the Deployment model and the BPMN model

some examples of security rules. A detailed description and implementation of these security rules can be found on the project's website.

Openflexo enables us to create behaviors that are used to express security rules using the FML language. FML is a domain-specific object-oriented language with syntax close to Java and an equivalent expressivity. In our interaction model, each concept that represents a correspondence between models contains a set of security rules. These rules are evaluated when the model elements linked by a concept change. We detail below three of such security rules:

- *SecurityRule-1*: when two pools in the BPMN model (that correspond to artifacts in the deployment model), communicate security-critical data and are deployed in different devices, the communication between the two devices must occur through an encrypted channel. This rule is added to the concept PoolArtifactCorrespondence. Note that this rule uses *SecurityRule-2*.
- *SecurityRule-2*: if a MessageFlow in the BPMN model has a security annotation, the corresponding CommunicationPath in the deployment model must also have a matching (*i.e.*, semantically equivalent) security annotation and vice versa. This rule is added to the concept MessageFlowCommunicationCorrespondence.
- *SecurityRule-3*: each user in the defined data model must have an associated role in the Access Control model. This rule is added to the concept UserRoleCorrespondence.

Let's start by the easier Listing 1. It contains a part of the implementation of the concept MessageFlowCommunicationPathCorrespondence and its associated behavior `iTrustSecurityRule2`. It implements *SecurityRule-2* by reporting a security rule violation if `verifySecureCommunication` it is not verified. This method ensures that both the MessageFlow and CommunicationPath possess appropriate security annotations (here encrypted for CommunicationPath and non-repudiation for MessageFlow). This verification ensures compliance with integrity and non-repudiation aspects when transmitting personal data.

```
public concept MessageFlowCommunicationPathCorrespondence {
  BPMNMessageFlow messageFlow with EMFObjectRole(container=bpmnModelAccess,type=
      BPMN_MESSAGE_FLOW);
  DeploymentCommunicationPath communicationPath with EMFObjectRole(container=
      deploymentModelAccess,type=DEPLOYMENT_COMMUNICATION_PATH);
  ...
  public void iTrustSecurityRule2() {
    boolean checkAnnotation = this.verifySecureCommunication();
    if (!checkAnnotation)
      log "[iTrust - SecurityRule 2] -> Security rule violated";
    ...
  }
  private boolean verifySecureCommunication() {
    boolean check = false;
    // Verify if the MessageFlow has a security annotations
    for (BPMNAssociation asso: select BPMNAssociation from bpmnModelAccess) {
      if (asso.targetRef == messageFlow)) {
        for (BPMNTextAnnotation annot: select BPMNTextAnnotation from
       bpmnModelAccess) {
          if (asso.sourceRef == annot) {
          // Verify if the CommunicationPath has a encrypted security annotation
            if (communicationPath.encrypted != null && (annot.text.contains("
       NonRepudMF") || annot.text.contains("ConfidentialityMF") || annot.text.
       contains("IntegrityMF")))
              check = true;
          }
      ...
    return check;
  }
}
```

Listing 1: iTrust - SecurityRule-2

In Listing 2, we present an excerpt of the implementation of the *SecurityRule-1* by the behavior `securityRule1` of the concept `PoolArtifactCorrespondence`. First, the rule verifies whether there exists a DataObject containing a confidentiality, integrity or privacy annotation between the flow elements belonging to the Pool. If such a DataObject exists, the behavior then proceeds to call `verifyExistenceConsistentInstance` to examine whether the two artifacts corresponding to the two Pools are deployed on different devices, and, if so, it verifies whether the communication between them occurs through an encrypted channel. To accomplish this, it calls other behaviors:

- `verifyExistenceArtifactAssociated` returns the list of artifacts associated with the artifact specified as parameter in the deployment model.
- `verifyExistenceCorresp` returns a list of artifacts that are a part of the instance of the ITrustSecurityFederation model.
- `getPool` selects the corresponding pool in the BPMN model for each selected associated artifact.
- `getDeployementDevice` returns the devices in the Deployment model where the artifact is deployed.
- `getCoomunicationPath` returns the communicationPath linking the device specified in the parameters with other devices.
- `compareDevices` compares the devices provided as parameters.
- `getSharedMessageFlows` returns a list of MessageFlows exchanged between the two selected pools in the BPMN model.
- `verifySecureCommunication` evaluates the *SecurityRule-2*. It checks whether any of the shared MessageFlows obtained through `getSharedMessageFlows` contain instances of `MessageFlowCommuncationPathCorrespondence`.

```
public concept PoolArtifactCorrespondence {
  BPMNParticipant pool ...;
  DeploymentArtifact artifact ...;
  ...
  public void securityRule1() {
    boolean checkConfidentialData = false;
    // Get the MessageFlow for the Pool in the BPMN model
    flowElts = container.getFlowElementsOfPool(pool);
    // Verify if exists a confidential dataObjects among the flow elements
    checkConfidentialData = container.verifyDataObjectAnnotation(flowElts);
    // This security rule request to verify the securityRule2
    if (checkConfidentialData == true && !this.verifyExistenceConsistentInstance
       ())
      log "[iTrust - SecurityRule 1] -> Security rule violated" ;
      ...
  }
  public boolean verifyExistenceConsistentInstance() {
    // Get associated artifacts
    List<DeploymentArtifact> assoArtifacts = container.
       verifyExistenceArtifactAssociated(artifact);
    List<DeploymentArtifact> assoArtifactsFilter = container.
       verifyExistenceCorresp(assoArtifacts);
    // Get Devices where the Artifact is deployed
    List<EMFObjectIndividual> devices = container.getDeploymentArtifact(artifact
       );
    // Get the communicationPath of the devices
    List<DeploymentCommunicationPath> commPath = container.getCommunicationPath(
       devices);
    // Get MessageFlow outgoing from Pool to the associated Pool
    for (DeploymentArtifact assoArtifactFilt : assoArtifactsFilter)
      BPMNParticipant associatedPool = container.getPool(assoArtifactFilt);
      List<EMFObjectIndividual>  devicesArtifactsFilter = container.
       getDeploymentArtifact(assoArtifactFilt);
      // Verify if the devices are deployed in the same node
      if (!container.compareDevices(devices,devicesArtifactsFilter))
      // Get the MessageFlow from the pool
        List<BPMNMessageFlow> messageFlowsPool = container.getMessageFlows(pool)
      ;
        List<BPMNMessageFlow> messageFlowsAssociatedPool = container.
      getMessageFlows(associatedPool);

        List<BPMNMessageFlow> sharedMessageFlows = container.
      getSharedMessageFlows(messageFlowsPool,messageFlowsAssociatedPool);
        for (BPMNMessageFlow dataflow : sharedMessageFlows)
          // VerifySecureCommunication --> SecurityRule2
          for (MessageFlowCommunicationPathCorrespondence item: container.
      listInstancesMessageComm) {
            if (dataflow == item.messageFlow && commPath.contains(item.
      communicationPath))
              if (!item.verifySecureCommunication())
                return false;
    return true;
  }
}
```

Listing 2: iTrust - SecurityRule-1

4.1.1 Evolution scenarios: Once security rules are attached to the correspondences, our *security federation* is ready to be used. This means that we can modify any of the federated models and get an automatic evaluation of its consistency regarding security, as shown in Figure 3 of Section 3. In Openflexo, a behavior defined on the federation itself is in charge of receiving and analyzing model changes to trigger the corresponding security rules. We illustrate this usage with two evolution scenarios:

(1) *Evolution scenario 1: Moving an artifact from a trusted device to a non-trusted one.* The change occurs in the Deployment model and consists of redeploying the Doctor artifact in the MobileDevice. This change triggers a change analysis process that effectively determines which element has changed and identifies the impacted correspondences. In the iTrustSecurityFederation, this process identifies the PoolArtifactCorrespondence instances to which the artifact Doctor is connected. Then, the behaviors attached to this concept and representing security consistency rules are executed. Concretely, the behavior corresponding to *SecurityRule-1* is launched. Additionally, the behavior requests to evaluate *SecurityRule-2* because *correspondence1* depends on *correspondence3*.
The result of the analysis of the changes indicates that both security rules are violated. Indeed, the BPMN model shows that the pool Doctor communicates with the pool iTrust to recover a prescription which is annotated with Integrity. However, the artifacts Doctor and iTrust may be deployed in different devices, and the communication path between these two devices is annotated Internet, which may violate the integrity of the prescription.

(2) *Evolution scenario 2 : Adding a new user.* In this scenario, a new user is added to the data model without being associated with a role.
As in the previous evolution scenario, the first step is a change analysis process which determines that in our security federation we need to deal with the concept UserRoleCorrespondence. This concept contains a behavior that evaluates the security consistency rule *SecurityRule-3* (note that this rule considers inheritance for the verification of assigned roles). If the added user does not inherit from an existing one, a security consistency error is triggered, as no role is assigned.

4.2 Holiday Booking

Holiday booking is a reservation system for travel agencies to book flights and hotels for its clients. This use case is taken from [2] in which the authors follow an MDA (Model-Driven Architecture) forward engineering approach for the integration of non-functional requirements (including security) in SOA (Service Oriented Architecture) and MDE. Concretely, the use case is made up of three models, a BPMN model representing travel agency processes as

a CIM (Computation Independent Model), a SOAML (SOA Modeling Language) model as a PIM (Platform Independent Model) and a WDSL model as a PSM (Platform Specific Model). All the models carry security information.

In this use case the SOAML model is derived from the BPMN model, and the WSDL model from the SOAML model by using model transformations. We reuse these model transformations to extract the matching rules required by our methodology. Concretely, we have used the following matching rules:

- *MatchingRule-1* A Pool in the BPMN model may correspond to a Participant in the SOAML model.
- *MatchingRule-2* A DataObject in the BPMN model may correspond to a Message in the WSDL model.
- *MatchingRule-3* An Interface in SOAML may correspond to an Interface in the WSDL model.
- *MatchingRule-4* An Operation in SOAML may correspond to an Operation in the WSDL model.

Figure 7 shows excerpts of the Holiday Booking use case models and the correspondences established using the matching rules aforementioned.

As in the previously discussed use case, once the models and the matching rules are available the *security federation* is created. First, we create a *Virtual Model* called HolidayBookingModelFederation. Next, we define a set of FlexoConcepts to represent the inter model correspondences (*e.g.*, concept DataObjectMessageCorrespondence represents the correspondence between a DataObject element in the BPMN model and a Message element in the WSDL model) together with the corresponding FlexoProperties using model slots defined to connect with each external model. Ultimately, we define eight security rules tailored for Holiday Booking, aimed to assess the integrity, access control, non-repudiation, and privacy aspects. In contrast to the iTrust case study, we define fewer security rules. This variation can be attributed to several factors, including the security information available in the models, the number and the kinds of models involved. Among the defined security rules, we described the security rules below and attached them to the adequate concept.

- *SecurityRule-1*: If a message in an operation in the WSDL model holds a correspondence with a DataObject in the BMPN model tagged with the integrity annotation, then, the binding parameter of the WSDL operation must be one with an encryption mechanism.
- *SecurityRule-2*: Each Participant element in the SOAML model associated with a Pool in the BPMN model must have compatible permissions.
- *SecurityRule-3*: Every Operation in the SOAML model must have a corresponding Operation in the WSDL model.

The provided FML code excerpt for HolidayBookingSecurityFederation, given in Listing 3, shows a part of the implementation concerning the concept of `DataObjectMessageCorrespondence` and the behavior `checkSecurityDataTransmission` which attempts to verify the two parts of the *SecurityRule-1* by invoking two distinct methods: 1) `checkDataObjectSecurityAnnotations` is called to determine whether the DataObject in the BPMN model has an Integrity annotation. 2) `checkFaultBinding` invoked to assess whether the Binding parameters of the WSDL operation, associated with the message given as parameter, incorporate a secure mechanism. This security rule verifies the consistency of the security concern, guaranteeing that confidentiality and integrity aspects are adequately addressed and maintained in both the BPMN and WSDL models.

```
public concept DataObjectMessageCorrespondence {
  BPMNDataObject dataObject with EMFObjectRole(container=bpmnModelAccess,type=
      BPMN_DATAOBJECT);
  WSDLElement message with EMFObjectRole(container= wsdlModelAcess, type =
      WSDL_ELEMENT);
  ...
  public void checkSecurityDataTransmission() {
    // verify the DataObject annotations (Non-repudiation, integrity)
    boolean checkAnnotation = container.checkDataObjectSecurityAnnotations(
      dataObject);
     // verify the Binding protocol of the operation holding message
    boolean checkBinding = container.checkFaultBinding(message);
    if ( checkAnnotation && !checkBinding )
      log "[HolidayBooking - SecurityRule 1] -> Security rule violated";
    ...
  }
}
```

Listing 3: HolidayBooking - SecurityRule-1

4.2.1 Evolution scenario: As with the previous use case, we illustrate the usage of the Holiday Booking Security Federation with an evolution scenario: *Adding a new operation, RequestBookTicket, in the WSDL model.* The RequestBookTicket operation has two parameters: ClientInfo which represents an input message and Binding Protocol an HTTP Binding protocol. The message parameter corresponds to the data object ClientInfo, considered private data in the BPMN model.

The analysis of this modification identifies that the DataObjectMessageCorrespondence is impacted. This identification is achieved by verifying the type of the changed element. In this evolution scenario, the parameter of the Operation corresponds to a Message type. Then, the verification of the instances of this correspondence determines which of the two cases described below is executed: if the message ClientInfo is part of the DataObjectMessageCorrespondence instance, *Case 2* is executed, otherwise, *Case 1* is executed.

- *Case 1*: this modification triggers first, the creation of a correspondence as a consequence of the creation of the message parameter (this is a semi-automatic step, as the user must validate that the correspondence is correct). Then the security rule attached to the concept DataObjectMessageCorrespondence is also activated. If the operation containing the message parameter declares as second parameter an unsafe HTTP Binding protocol, a security inconsistency is detected and reported to the user.
- *Case 2*: the correspondence is established, the modification requires only the analysis of the security rule attached to the Concept DataObjectMessageCorrespondence.

Additionally, the change analysis process identifies that the Operation is a property of the concept OperationsInterfaceCorrespondence in the HolidayBookingSecurityFederation and the behavior attached to this concept is activated. Concretely, the *SecurityRule-3* that verifies that each operation in the SOAML model corresponds to an operation in the WSDL model. The result of the analysis reveals a violation of the security rule due to the addition of the RequestBookTicket operation in the WSDL model.

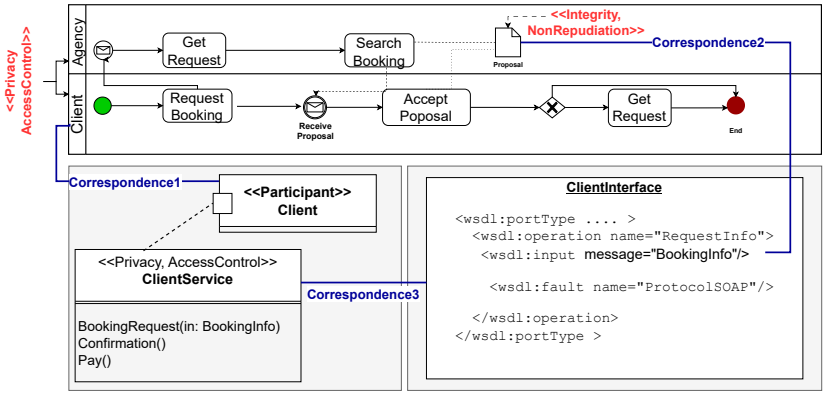

Figure 7: Excerpt of Holiday Booking models and the correspondences

4.3 Discussion

Preliminary results indicate that our methodology provides effective support to the consistency management of multi-model systems in different model evolution scenarios. However, the evaluation of our methodology is currently based on a limited number of case studies.

Compared to more classical approaches such as model integration, model unification and MDA forward engineering ours is more flexible and allows the evolution of any of the involved models in any direction while they remain in their technological spaces. As an example, MDA forward engineering (the approach followed by the original Holiday Booking use case) supports evolution in one direction by relaunched transformations and code generators. However, evolution in the reverse direction is not supported. Similarly, we may use UML enriched with security information and OCL (to encode security rules) to develop secured systems with support of evolution, but this imposes the use of UML even when it may not be the most appropriate language to describe certain aspects of the system. Future work will involve a systematic evaluation by referring to the descriptions provided in [27] using a broader range of case studies to confirm the effectiveness of our methodology in diverse and complex evolution scenarios.

The degree of *usefulness* of our methodology depends on the following factors: 1) enough security information should be available in the models participating in the federation; 2) security and modeling expertise must build a **good** *security federation*. This includes the identification of relevant correspondences and the encoding of requirements for security consistency management by using the model's security information in *security rules*. We think however that building *security federation* reifies implicit security knowledge and enables its reuse so that building *security federations* gets eased as the approach is adopted.

Our methodology does not prescribe a level of granularity for correspondences or *security rules*. In this sense, two styles (or a mix of the two) are possible when building a *security federation*: a more local approach uses fine-grained correspondences and more simple *security rules* that only use the information of the elements directly linked by a correspondence. A less local approach allows *security rules* to use more information. Global rules may be richer, but local rules support finer diagnostics. Determining which of the styles is more effective requires further research.

From the *security federation* usage perspective, when a model evolves and impacts security consistency, our methodology gives feedback. This feedback consists of the changed element(s) that triggered the re-evaluation of the *security rules* that do not hold anymore together with the rules themselves. The case studies done convinced us that this feedback is sufficient for restoring consistency (*e.g.*, by reverting a change) in many evolution scenarios.

5 CONCLUSIONS & FUTURE WORK

In this paper, we have presented a novel approach to tackle the problem of security consistency in multi-modeling. We have done so by leveraging model federation. Concretely, we have presented a methodology to build what we call *security federations* in which the security dependencies between models are reified and equipped with *security rules*. We have applied this methodology to several use cases, demonstrating its feasibility and its ability to detect inconsistencies due to different evolution scenarios.

As a future work we intend to explore the following research directions: first, we intend to further evaluate the usability aspects of our methodology and tooling and explore the potential for automation of some of our steps (*e.g.*, the initial matching of security concepts); second, we plan to extend our methodology to other phases of the development process (*e.g.*, by including other artifacts such as configuration scripts of deployed systems); finally, we envisage the integration of (semi) automatic model repair approaches as a complement to our methodology.

Acknowledgements:

The authors acknowledge the ANR (French National Research Agency) for its financial support of the MODES project n° ANR-23-CE25-0011-01, This work is also partially funded by Région Bretagne.

REFERENCES

[1] Artur Boronat, Alexander Knapp, José Meseguer, and Martin Wirsing. 2009. What is a multi-modeling language?. In *Recent Trends in Algebraic Development Techniques: 19th International Workshop, WADT 2008, Pisa, Italy, June 13-16, 2008, Revised Selected Papers 19*. Springer, 71–87.

[2] Abdelhadi Bouain, Abdelaziz El Fazziki, and Mohammed Sadgal. 2014. Integration of non-functional requirements in a service-oriented and model-driven approach. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 1–8.

[3] Chahrazed Boudjemila, Fabien Dagnat, and Salvador Martínez. 2023. Towards evolving secured multi-model systems with model federation. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 939–943.

[4] Jens Bürger, Stefan Gärtner, Thomas Ruhroth, Johannes Zweihoff, Jan Jürjens, and Kurt Schneider. 2015. Restoring security of long-living systems by co-evolution. In *COMPSAC 2015*, Vol. 2. IEEE, 153–158.

[5] Jens Bürger, Daniel Strüber, Stefan Gärtner, Thomas Ruhroth, Jan Jürjens, and Kurt Schneider. 2018. A framework for semi-automated co-evolution of security knowledge and system models. *Journal of Systems and Software* 139 (2018), 142–160.

[6] Mahmoud El Hamlaoui, Saloua Bennani, Mahmoud Nassar, Sophie Ebersold, and Bernard Coulette. 2018. A MDE Approach for Heterogeneous Models Consistency.. In *ENASE*. 180–191.

[7] Mahmoud El Hamlaoui, Sophie Ebersold, Saloua Bennani, Adil Anwar, Taoufiq Dkaki, Mahmoud Nassar, and Bernard Coulette. 2021. A Model-Driven Approach to align Heterogeneous Models of a Complex System. *The Journal of Object Technology* 20, 2 (2021), 1–24.

[8] Johannes Geismann, Christopher Gerking, and Eric Bodden. 2018. Towards ensuring security by design in cyber-physical systems engineering processes. In *Proceedings of the 2018 international conference on software and system process*. 123–127.

[9] Fahad R Golra, Antoine Beugnard, Fabien Dagnat, Sylvain Guerin, and Christophe Guychard. 2016. Addressing modularity for heterogeneous multi-model systems using model federation. In *Companion Proceedings of the 15th International Conference on Modularity*. 206–211.

[10] Torben Mejlvang Hangensen and Bent Bruun Kristensen. 1992. Consistency in software system development: Framework, model, techniques & tools. *ACM SIGSOFT Software Engineering Notes* 17, 5 (1992), 58–67.

[11] Chadni Islam, Muhammad Ali Babar, and Surya Nepal. 2019. An ontology-driven approach to automating the process of integrating security software systems. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. IEEE, 54–63.

[12] Jan Jürjens. 2002. UMLsec: Extending UML for secure systems development. In *International Conference on The Unified Modeling Language*. Springer, 412–425.

[13] Heiko Klare and Joshua Gleitze. 2019. Commonalities for preserving consistency of multiple models. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 371–378.

[14] Dimitrios Kolovos, Richard Paige, and Fiona Polack. 2008. Detecting and repairing inconsistencies across heterogeneous models. In *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 356–364.

[15] Kurt Kosanke. 2006. ISO Standards for Interoperability: a comparison. In *Interoperability of enterprise software and applications*. Springer, 55–64.

[16] Luciano Marchezan, Wesley KG Assuncao, Roland Kretschmer, and Alexander Egyed. 2022. Change-oriented repair propagation. In *Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering*. 82–92.

[17] Andrew Meneely, Ben Smith, and Laurie Williams. 2012. Appendix B: iTrust electronic health care system case study. *Software and Systems Traceability* (2012), 425.

[18] Saraju P Mohanty. 2020. Security and Privacy by Design is Key in the Internet of Everything (IoE) Era. *IEEE Consumer Electron. Mag.* 9, 2 (2020), 4–5.

[19] Sven Peldszus, Jens Bürger, Timo Kehrer, and Jan Jürjens. 2021. Ontology-driven evolution of software security. *Data & Knowledge Engineering* 134 (2021), 101907.

[20] Sven Matthias Peldszus. 2022. *Security Compliance in Model-driven Development of Software Systems in Presence of Long-Term Evolution and Variants*. Springer Nature.

[21] Qusai Ramadan, Mattia Salnitriy, Daniel Strüber, Jan Jürjens, and Paolo Giorgini. 2017. From secure business process modeling to design-level security verification. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 123–133.

[22] Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini. 2017. Designing secure business processes with SecBPMN. *Software & Systems Modeling* 16 (2017), 737–757.

[23] Patrick Stünkel, Harald König, Yngve Lamo, and Adrian Rutle. 2021. Comprehensive systems: a formal foundation for multi-model consistency management. *Formal Aspects of Computing* 33, 6 (2021), 1067–1114.

[24] Patrick Stünkel, Harald König, Adrian Rutle, and Yngve Lamo. 2021. Multi-model evolution through model repair. (2021).

[25] Weslley Torres, Mark GJ Van den Brand, and Alexander Serebrenik. 2020. A systematic literature review of cross-domain model consistency checking by model management tools. *Software and Systems Modeling* (2020), 1–20.

[26] Burak Ünver and Ricardo Britto. 2023. Automatic Detection of Security Deficiencies and Refactoring Advises for Microservices. In *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. IEEE, 25–34.

[27] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.