Towards Automated Attack Simulations of BPMN-based Processes

Simon Hacks
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
Odense, Denmark
shacks@mmmi.sdu.dk

Robert Lagerström
Division of Network and Systems Engineering
KTH Royal Institute of Technology
Stockholm, Sweden
robertl@kth.se

Daniel Ritter
SAP SE
Walldorf (Baden), Germany
daniel.ritter@sap.com

***Abstract*—Process digitization and integration is an increasing need for enterprises, while cyber-attacks denote a growing threat. Using the Business Process Model and Notation (BPMN) is common to handle the digital and integration focus within and across organizations. In other parts of the same companies, threat modeling and attack graphs are used for analyzing the security posture and resilience.**

In this paper, we propose a novel approach to use attack graph simulations on processes represented in BPMN. Our contributions are the identification of BPMN's attack surface, a mapping of BPMN elements to concepts in a Meta Attack Language (MAL)-based Domain-Specific Language (DSL), called coreLang, and a prototype to demonstrate our approach in a case study using a real-world invoice integration process. The study shows that non-invasively enriching BPMN instances with cybersecurity analysis through attack graphs is possible without much human expert input. The resulting insights into potential vulnerabilities could be beneficial for the process modelers.

***Index Terms*—Attack Simulations, BPMN, Integration Processes, Meta-Attack Language, Threat Modeling**

I. Introduction

In a highly connected world, in which enterprises get more and more intertwined with each other, digitization drives the change of our society, which enables innovation, increased connectivity, improved productivity, and more accessible information [13]. Many organizations move their applications into the cloud, integrate them within the organization and across different organizations [25]. To facilitate the related internal and inter-organizational business and application integration processes, many organizations use the Business Process Model and Notation (BPMN) [23], [24], [26]. At the same time, we perceive a significant growth of cyber-attacks [12], [20]. A potential approach to address these security threats is an assessment of integrated (business) processes from a cyber security perspective.

Different ways to enrich the standardized BPMN with new concepts that reflect various security aspects already exist (e. g., [16], [19], [22], [28], [34]). However, if organizations have a large set of productive processes, adding supplementary information is often an unfeasible effort. Moreover, those processes are often created by business experts and process consultants, who regularly have no security expertise. Even when assuming that process-aware runtime systems are routinely assessed regarding potential security threats, scenario-specific configurations, and design decisions (e. g., selection of server or library versions) could make instantiated processes vulnerable to attacks. Finding all possible vulnerabilities and avoiding exploits is extremely difficult for process modelers and project consultants. At the same time, exploits could have a huge impact across overlapping or connected processes, and thus endanger larger parts of a business.

For example, the BPMN-based integration process in Ex. 1 exhibits several points of attack that might not be obvious during its scenario-specific instantiation.

Example 1. Figure 1 shows a simplified version of an invoicing integration process for Italian companies. An ERP system sends invoices to the integration process, which **1** preserves information like the transfer *mode*, as well as relevant identifiers like *IdentityCode* (data exchange not shown), and then signs the invoice itself. In **2** a Sistema di Interscambio (SdI)[1] compliant message is prepared and the signed invoice is cached, before storing it for further reference. Message headers required by SdI are set and **3** either sent to a test or production endpoint, else discarded, depending on its mode. The response message is enriched with information preserved from the original message **4**, stored for reference and then **5** a compliant response message for the ERP system is prepared. ■

Starting with the data transfer from the ERP system to the integration process in the form of a message, the remote call is conducted over public networks to a scenario-specific endpoint configuration (e. g., HTTPS server). Such configurations are prone to security vulnerabilities 🛡, potentially leading to an exploit ☣ of the integration process runtime. Similarly, subsequent script steps allow for user-defined source code that could exploit vulnerabilities of the script engines, and consequently the integration runtime. Service tasks accessing an associated, global data store (not shown), as well as remote calls to the SdI endpoints could lead to attacks on the connected database, remote server endpoints, respectively. Finally, the

[1]SdI is an Italian platform for issuing, transmitting, and saving invoices.

978-1-6654-3579-6/21/$31.00 ©2021 IEEE

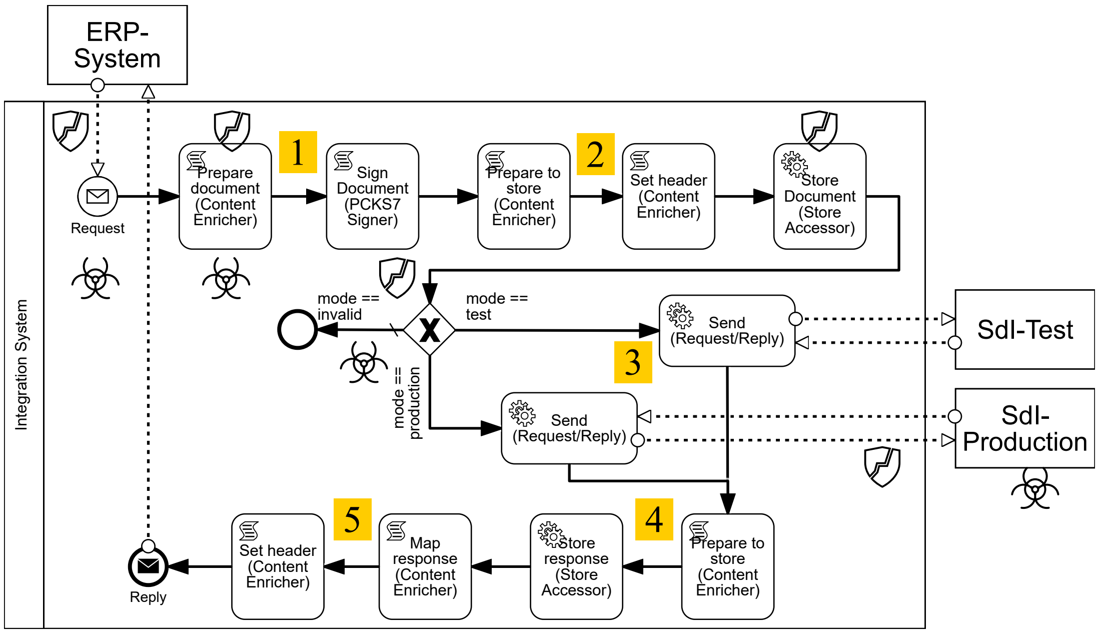
ERP-System
Integration System
Request
Prepare document (Content Enricher)
1
Sign Document (PCKS7 Signer)
Prepare to store (Content Enricher)
2
Set header (Content Enricher)
Store Document (Store Accessor)
mode == invalid
mode == test
mode == production
Send (Request/Reply)
3
Send (Request/Reply)
SdI-Test
SdI-Production
Prepare to store (Content Enricher)
4
Store response (Store Accessor)
Map response (Content Enricher)
5
Set header (Content Enricher)
Reply

Fig. 1. SAP CPI eDocuments Italy invoicing integration process (simplified from [30]) with non-mandatory, explanatory vulnerabilities and exploits

user-defined conditions in the exclusive gateway could exploit weaknesses in the used parser.

We aim to develop a solution that allows to assess the security state of BPMN processes without the need to edit the process models. To perform such a security assessment, one has to solve the challenging tasks of identifying vulnerabilities, understanding security-relevant parts, and determining potential attacks [21]. The use of attack simulations based on system architecture models have been proposed for addressing these challenges, (e. g., [5], [11]). These approaches have in common that they rely on a static implementation of the used meta-model. Therefore, the use of the Meta Attack Language (MAL) [14] was proposed. MAL is a framework to create domain-specific languages (DSLs) that defines the generic attack logic codified in languages such as coreLang [15] or powerLang [9].

To assess the security state of business or integration processes, we map BPMN to coreLang (a DSL representing general IT concepts) [15]. Based on this mapping, we automatically transform the concrete process instances into attack models that can be simulated in an attack simulation tool called securiCAD [5]. As a consequence, our approach is fully model-driven and incorporates implementation details, but does not include implicit knowledge (e. g., hidden in ERP systems). We refer to process mining [32] for generating models and providing the needed implementation details of processes without an explicit process model. Subsequently, we elaborate more detailed on our applied research method:

Methodology In this work, we design an artefact that provides a security assessment for information systems. Accordingly, we rely on the principle of Design Science Research (DSR) [10]. To decide which concrete implementation of DSR we apply, we follow Venable et al. [33], who sketch a decision support that helps to choose the best fitting approach. As we are focusing on just one organization at the moment, we opt for Action Design Research (ADR) [31], which is comprised of four stages, which are illustrated subsequently.

Stage 1: Problem Formulation We face the challenge of large repositories of business and application integration processes between different organizations. To reduce the risk of unwanted compromise of the connected systems, we want to assess the actual security state within these processes. Usually, one would scan the systems and perform an automated vulnerability analysis, which is not possible because the processes cover systems of different organizations and vulnerabilities could be introduced in scenario-specific configurations.

Due to the size of the repository, it is also not feasible to follow other approaches that extend BPMN with security related concepts, because it would be too much effort and the process owners might lack the needed security knowledge.

To sum up, we face following challenges in our research:

- A large set that hinders (a) a manual security assessment and (b) the reuse of solutions that expect complementing the documented processes.
- The processes involve different organizations and scenario-specific configurations. Hence, conventional network scans are not an option.
- The process owners might have insufficient security background / knowledge.

From these challenges, we deduct the following objectives:

(i) **Processes:** The solution should be able to assess the security of processes notated with BPMN.
(ii) **Automatic:** The assessment should be automatic (i. e., to account for a potentially high number of processes).

(iii) **Non-Invasive:** The solution should not demand changes to the existing process documentation.

(iv) **Simulation:** The solution should simulate different applications that are hidden behind interfaces.

Stage 2: Building, Intervention, and Evaluation In stage 2, we follow an IT-dominant approach. On the one hand, we have a team of researchers who develop the artefact presented in Sect. IV, while a practitioner continuously provides his feedback on the artefact. The end-users are representatives of the industrial partner developing the processes and others responsible for the security.

Stage 3: Reflection and Learning While in stage 2 a specific solution for a certain problem is developed, stage 3 focuses on the continuous interchange between the researchers and the practitioners. Therefore, we have set up regular meetings in which the recent developments were discussed and mirrored against the industrial partner's reality.

Stage 4: Formalization of Learning The outcomes of stage 2 denote the following contributions of this work:

- a BPMN process security classification (focus on process and inter-process, but not conversations),
- a mapping for automated translation of BPMN to coreLang, which covers 53.8 % of relevant[2] BPMN concepts and 73.3 % the coreLang concepts,
- and a prototype for attack simulations.

Outline We set our approach into context to related work in Sect. II and give sufficient background in Sect. III, before we specify a mapping of BPMN to coreLang in Sect. IV. We evaluate the mapping in Sect. V and conclude in Sect. VI.

II. Related work

In this section, we discuss related work in the intersection of BPMN and security in the context of our objectives (i)–(iv), summarized in Tab. I. Essentially, we found that (a) most of the current approaches require meta-model extensions that add a considerable number of security-related BPMN components, of which many are (b) excessively verbose, and thus difficult to manage (e. g., [22], [28], [34]) or fail to express security concepts in a format that is fully comprehensible to business experts, others propose only a theoretical or descriptive extension (e. g., [17], [19]).

Threat modeling A threat profile security framework was proposed as a BPMN extension by Zareen et al. [34]. The authors leveraged the extension mechanism provided in BPMN 2.0 to model threat-based security requirements and introduced several graphical components for BPMN diagrams.

Meland et al. [19] related the concept of threat modeling to the concept of business process modeling, presenting four different ways for threat specification at designing-time within BPMN. In particular, they discuss the pros and cons of threat representation as error events, escalation events, annotations, and through meta-model extensions.

[2]We solely focus on process artifacts (i. e., no collaboration artifacts and only those choreography artifacts that are used within processes) and within a process we do not consider visual concepts like grouping or comments.

TABLE I
BPMN2MAL IN THE CONTEXT OF RELATED WORK

Literature / objective	Processes (cf. (i))	Automatic (cf. (ii))	Non-Invasive (cf. (iii))	Attack simulation (cf. (iv))
Threat modeling	👍	🖓	🖓	🖓
Security modeling	👍	🖓	🖒	🖓
Goal modeling	👍	🖓	🖓	🖓
BPMN2MAL	👍	🖒	👍	👍

👍: supported, 🖒: partially supported, 🖓: not supported, ␣: n/a

The found threat modeling approach target objective (i), but do not fulfill objectives (ii–iv)).

Security modeling Rodriguez et al. [27] propose a non-compliant BPMN meta-model extension including predefined set of high-level cybersecurity requirements (e. g., privacy, integrity) into BPMN process diagrams (mainly pool, message flow, data object and activity), enabling business analysts to express their security needs. The approach was studied for a healthcare process. Similarly, Mülle et al. [22] specify a constraint language to formulate security attached via text annotations expressing high-level security requirements. Consequently, the diagrams become complex and expert-oriented, and thus business experts could find it hard to understand. Sang et al. [29] also extended the BPMN meta-model with new security elements, which can be also represented within the BPMN diagrams.

A meta-model extension introduced by Brucker et al. [1] with regards to privacy requirements like access control, separation of duties, binding of duty, and need to know. Beyond the SecureBPMN language, the authors propose a tool to model the security concepts during the modeling phase and also to enforce them at runtime. Cherdantseva et al. [3] extended BPMN to include Information Assurance & Security (IAS) requirements (incl. vulnerabilities, threats, and risks). Most notably, the work enriches BPMN data-related elements like Data Object as IAS Asset and Message Flow as Vulnerability or Risk.

Chergui et al. [4] proposed a BPMN meta-model extension based on the security requirements derived from the cybersecurity ontology [19]. The extension is fully BPMN compliant and, in contrast to the other works, leverages the BPMN meta-model extension mechanism introduced in BPMN version 2.0. Also, a web tool was proposed to facilitate collaboration between business and security experts and provided an XML schema extension for integration with the existing BPMN modeler tools.

Maines et al. [16] introduced a comprehensive cybersecurity ontology for specifying security requirements within BPMN, identifying a total of 79 security concepts. It was later extended [17] to represent the BPMN security requirements in a third dimension. However, the extension is described only theoretically.

The security modeling approaches target objective (i) and partially support (iii), but violate objectives (ii) and (iv).

Security goal modeling Salitri et al. [28] proposed a framework to express security requirements in terms of BPMN annotations, called SecBPMN. SecBPMN allows for the description of system information, whereas the security policies are defined through SecBPMN-Q, a query language for BPMN. The annotated security requirements are derived from the Reference Model of Information Assurance and Security [2] and include accountability, auditability, authenticity, availability, confidentiality, integrity, non-repudiation, and privacy.

The goal modeling approach targets objective (i), but does not fulfill objectives (ii–iv)).

BPMN2MAL We focus on a non-invasive modeling of BPMN-based processes without additional security artefacts in BPMN. Instead BPMN2MAL strives to automatically map and conduct attack simulations on existing BPMN diagrams and their runtime configurations (e. g., remote endpoint interfaces and operations, or scripts and engine configurations).

III. Background

In this section, we give an introduction to MAL, which is the framework in which `coreLang` is created. `coreLang` serves as goal for our mapping from BPMN as it is a general representation of IT-dependent systems [15].

A. Meta Attack Language

MAL is a language framework that combines probabilistic attack and defense graphs with object-oriented modeling. It can be used for creating Domain-Specific Languages (DSLs). Such a language defines what information is required and specifies the generic attack logic about the domain studied. We refer to the original paper [14] for a detailed overview of MAL.

To create a MAL-based language, the first thing is to identify all relevant assets and their associations within a particular domain. Each asset can contain multiple attack steps, representing real threats. One compromised attack step can lead to (represented by "->") a next attack step, where each attack step is of the type OR (represented by "|") or AND (represented by "&"). OR indicates that an attacker can work on this attack step as soon as one of its parent attack steps is compromised, while AND indicates that all its parent attack steps must be compromised for an attacker to reach this step. An asset may also feature defenses (represented by "#"). The sum of attack paths is the attack/defense graph used for the attack simulation. Also, assets can inherit from each other, which means that an inherited asset inherits all attack steps and defenses of its parent asset (unless explicitly stated otherwise).

In Listing 1, we present a short example of a MAL-based DSL. It contains four modeled assets, the attack step connections, and the connections between assets. In Line 7, the *connect* attack step is of type OR, while in Line 15 is an AND attack step. The -> symbol denotes the connected next attack step. Thus, a *phish* on the `User` leads to *obtain* on the associated `Password`, and finally to *authenticate* on the associated `Application`. In Lines 28 to 32, the `associations` between the assets are defined.

```
category System {                                                    1
  asset Connection {                                                 2
    | access                                                         3
      -> app.connect                                                 4
  }                                                                  5
  asset Application {                                                6
    | connect                                                        7
      -> access                                                      8
    | authenticate                                                   9
      -> access                                                      10
    | guessPwd                                                       11
      -> guessedPwd                                                  12
    | guessedPwd [Exp(0.02)]                                         13
      -> authenticate                                                14
  & access                                                           15
  }                                                                  16
  asset User {                                                       17
    | attemptPhishing                                                18
      -> phish                                                       19
    | phish [Exp(0.1)]                                               20
      -> pwds.obtain                                                 21
  }                                                                  22
  asset Password extends Data {                                      23
    | obtain                                                         24
      -> app.authenticate                                            25
  }                                                                  26
}                                                                    27
associations {                                                       28
  Connection[con] * <- Acc -> * [app]Application                     29
  Application[app] 1 <- Cred -> * [pwds]Password                     30
  User[user] 1 <- Cred -> * [pwds]Password                           31
}                                                                    32
```

Listing 1: Example MAL Code

B. coreLang

As illustrated before, MAL provides the basics to create a threat modeling language from scratch. However, many languages created with MAL share a common set of concepts. To reduce unnecessary redundant work, `coreLang` [15] was developed. coreLang is comprised of predefined assets that appeared in different languages created with MAL. Thus, this coreLang can serve as starting point to model more domain specific languages or even act as a rudimentary language to model simple environments [8].

Fig. 2 presents the overall structure of coreLang. For coreLang, we have identified six different main categories: system, vulnerability, user, IAM (Identity and Access Management), data resources, and networking. Following, we discuss the concepts of coreLang, that we will use later on for our mapping. For more details, we refer to the original publication [15].

Often, attackers gather initial access to a system by exploiting human factors, represented in our case by the concept of `User`. These `User` interact with `Application` through one or several `Identity`, which codifies the access rights that the user has. To secure the use of the `Identity`, `Credentials` can be applied, which are classically stored as `Data` and, thus, can be for example encrypted.

Usually, `Data` is managed by an `Application`, which represent the main concept used in the following. An `Application` can run another `Application`, which can be used to model for example an operation system executing a software. Moreover, `Application` can communicate with each other via a `Connection` where they exchange `Data`.

In an `Application`, common attack steps are codified.

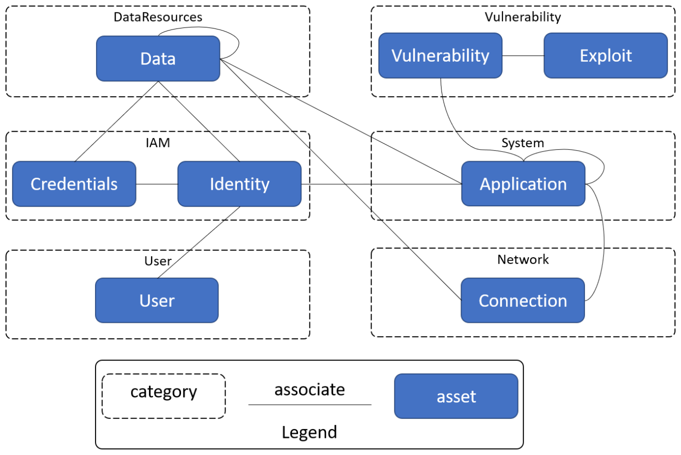

Fig. 2. Excerpt of `coreLang` containing used concepts (adapted from [15]).

However, if the end-user of `coreLang` is aware of a certain `Vulnerability`, which is not included in the common attack steps, they can use the concept of `Vulnerability` and `Exploit` to add it to a certain `Application`.

IV. BPMN Attack Surface and Mapping to coreLang

Hitherto, we have presented MAL and `coreLang`, which are thought to serve as means for the security analysis. Next, we will elaborate on possible elements of BPMN that can be facilitated as attack surface. Then, the identified surfaces are mapped to `coreLang`.

A. *Attack surface of BPMN*

Concepts like vulnerabilities and exploits as part of cyber security attacks captured by MAL and `coreLang` are crucial to be assessed in BPMN. We subsequently assess elements of BPMN that can be facilitated as attack surface due to their inherited properties (cf. Tab. II). We consider configuration properties like conditions, expressions, scripts, service calls, and data flow as points of attack. We do not further consider control flow elements as these are managed by the process engine and do not provide an attack surface by themselves. Moreover, we assume that the process engine as central component is security checked.

The first property of BPMN we shed light on are conditions. A condition is some kind of threshold that — if it crossed — triggers a certain behavior in the process. As thresholds are usually defined by the user, these can be manipulated by the attacker. Alternatively, it is also possible that the attacker changes the input for the condition to (not) trigger it intently. E.g., *Start Condition* is such an event as a process starts if a certain condition is fulfilled, which could have been manipulated by an attacker. Another example is a *Conditional Sequence Flow*, as those include a condition check before continuing, which can be manipulated by an attacker to cause undesired behavior of the process.

Similar to conditions are expressions, which can be more elaborated and are interpreted. Thus, expressions provide a

TABLE II
Relevant BPMN elements for cybersecurity attacks

BPMN	Sub	Condition	Expression	Service Call	Script	Collab. / Proc.	Data Flow
Data object							✓
Data Store							✓
Message							✓
Participant / Process	Pool, Lane					✓	
	Sub-Process					✓	
	Event Sub-Process					✓	
Event	Start						
	Message Start						✓
	Start Timer		✓				
	Start Error						
	Start Compensation						
	Start Parallel						
	Start Escalation						
	Start Condition	✓					
	Start Signal						
	Start Multiple						
	End						
	Message End	✓					✓
	End Error						
	Cancel End						
	End Compensation						
	End Signal						
	End Multiple						
	Terminate End						
Activity	User Task			✓			
	Script Task				✓		
	Service Task			✓			
	Send Task			✓			
	Receive Task			✓			
	Manual Task			✓			
	Business Rule Task		✓				
	Call Activity			✓			
Gateway	Exclusive						
	Event-based						
	Inclusive						
	Parallel						
	Complex	✓					
Sequence Flow	standard / default						
	conditional	✓					
Association							
Message Flow			✓				✓

gateway for attackers to manipulate a process to their demands. e. g., a *Business Rule Task* is defined by a complex expression that evaluates a set of input parameters and takes a decision based on this input (e. g., by a business rule engine). An attacker can either make changes to the input parameters or — if they want to create more permanent changes — manipulate the expressions themselves, and thus exploiting the business rule engine. Further, there are *Start Timer*, which can contain an expression that regulates when a process is triggered. Here, an attacker could manipulate the starting time to initiate the process more often/seldom as expected.

One main task of information systems is to process data.

Accordingly, this data is also of interest for attackers, who might want to get access to this data or even to manipulate it. The latter can cause severe issues for the process, as important decisions can be taken based on this data. Therefore, data related elements such as *Data object* or *Message* need special consideration in the security analysis.

Scripts are a powerful tool for developers to easily adopt requested functionality without the need for compiling new versions of programs. This made them very popular among process designers to alter processes to meet the customers' desires. However, a *Script Task* can serve also as attack surface, as the script might be prone to manipulations or the interpreter behind the script has unsolved vulnerabilities.

It is essential for integration processes to provide collaboration mechanisms. As those mechanisms are responsible for organizing the entire process, they need to be in contact with all related elements of the process. Consequently, mechanisms of collaboration are of tremendous interest for an attacker, as the mechanisms can bring the attacker in the position to take over the entire process. Classically, these mechanisms are presented by a *Participant* or a *Pool*. Alternatively, a *(Sub)Process* can also be such a means. If an attacker receives access to the application behind the orchestration, they will be in a good position to attack other related activities.

Finally, there are service calls, in which activities are performed outside of the integration process. As the services elaborating on the call can even be outside of the organization or desire human interaction, they are prone to be exploited by an attacker. For example, a *User Task* could be exploited by tricking employees to perform not-safe actions. Considering a *Service Task*, the service can run a outdated server version that will easily allow the attacker to take it over and use it as a starting point to penetrate the process.

For the later, we do not consider those BPMN elements in Tab. II, that did not receive a ✔, as relevant for our mapping. Either they model a process related behavior that is not of relevance for a static threat model (e. g., *Sequence Flow*) or we consider them as already implicitly included in other elements. An example for the latter is the gateway. An attacker could exploit it by manipulating its input parameters. However, the behavior of gateways is managed by the collaboration mechanisms and, thus, represent no additional threat to be considered for our attack simulations.

B. BPMN to coreLang Mapping

Before, we have determined the relevant elements of BPMN to be mapped to `coreLang`. Our mapping relies mainly on three mapping rules, that can be summarized as follows:

1) Data related BPMN elements will be mapped to `Data`.
2) Elements that require user interaction are mapped to `User`, `Identity` (the digital representation of the user), and `Credentials`.
3) Elements that can be configured or programmed by the process designer, are mapped to `Application` and `Connection`, where the latter illustrates the communication between several `Application`.

TABLE III
MAPPING FROM BPMN TO `coreLang`

BPMN	Sub	Data	Credentials	Identity	User	Application	Connection
Collaboration	Participant / Process					✔	✔
	Sub-Process					✔	✔
	Event Sub-Process					✔	✔
Event	Message Start	✔					
	Start Timer					✔	✔
	Start Condition					✔	✔
	Message End	✔					
Activity	User Task		✔	✔	✔		
	Script Task					✔	✔
	Service Task					✔	✔
	Send Task					✔	✔
	Receive Task					✔	✔
	Manual Task		✔	✔	✔		
	Business Rule Task					✔	✔
	Call Activity					✔	✔
Data	Data object	✔					
	Data Store					✔	✔
	Message	✔					
Connecting	Sequence Flow (conditional)					✔	✔
	Message Flow					✔	✔

However, these rules have exceptions, which we will discuss in the following.

In Tab. III, we present our mapping for the **collaboration** aspects of BPMN to `coreLang`. Our general interpretation of *Processes* is that they orchestrate the overall happening. Therefore, a tool is needed taking over this task such as a process engine. Based on this observation, we map *Process* and *Sub-Process* to an `Application`.

As MAL-based languages do not have a focus on the process flow but on the static connection between the different assets, there is generally no counterpart for the different **events** of BPMN in `coreLang`. However, there are some exceptions: If the event is related to a message (i. e., *Message Start* or *Message End*), then there is obviously a communication possible. This communication relates to an exchange of information, which can be manipulated. Therefore, we map *Message Start* and *Message End* to `Data`. Further, there are *Start Timer* and *Start Condition* that are triggered by an `Application` that continuously evaluates the underlying conditions.

Next, we discuss the mapping from **activities** to `coreLang`. Firstly, BPMN defines *User Task*, where a human is expected to interact with the process. These interactions are a classic attack surface and, thus, we map *User Task* to `User` in `coreLang`. Secondly, there are automated activities, such as *Script Task* or *Service Task*. We map both to `Application`, because a *Script Task* needs an engine (i. e., `Application`) executing a script, while a *Service Task* classically abstracts an interaction with e. g., a web-service, which are also executed by an `Application`.

Regarding the related `Connection`, we need to differentiate between synchronous and asynchronous calls. Basically, there is one `Connection` for synchronous calls, because the communication is initiated once. In contrast, there are two `Connection` for asynchronous calls, as communication gets initiated twice. Accordingly, *Send Task* and *Receive Task* have one `Connection`, while *Service Task*, *Message Start*, and *Message End* have one `Connection`.

BPMN is a process-oriented language. Hence, it provides **gateways** that describe branching points of the process. In contrast, MAL-based languages codify the static connection between the different assets. Consequently, there is no equivalent in `coreLang` and just the general connections between the linked activities are transferred.

Often, integration processes cope with **data** by transferring or transforming it. Moreover, to achieve access to data is often the ultimate goal of attackers. In BPMN this concept is modeled by *Data object* and *Message*. As there is no differentiation in `coreLang`, both are mapped to `Data`. If a *Data object* is stored outside of the process, then this is modeled by means of a *Data Store*, which can be for example a file system or a database. Obviously, this can be mapped to an *Application* (cf. Tab. III). For the threat modeling, flows are of no greater importance as indicated before. Nonetheless, there are exceptions in the **connecting** category of BPMN that require some logic and, thus, are mapped to `Application` (cf. Tab. III). In a *Conditional Sequence Flow*, the `Application` determines based on conditions if a communication takes place, while in a *Message Flow* expressions are evaluated.

V. Case Study

A. Bpmn2Mal *Prototype*

To demonstrate our approach, we implemented a prototype[3] (cf. Fig. 3), which has a process model in BPMN as central input. To maintain objective (iii) –non-invasive–, we do not directly elaborate on the process model, but transform the model to a graph representation. This representation includes the overall structure of the process and the attributes that are necessary to perform the mapping from BPMN to `coreLang`.

To meet objective (iv) –simulation–, which arises from the challenge that we do not know the concrete software specification beyond every interface, we create different graphs that are enriched with respective specifications. Therefore, we provide a list of configurations for each included BPMN element that contains the possibly used libraries and the expected versions. This list is then used to crawl the NIST NVD[4] for known vulnerabilities of these possible solutions (e. g., Apache web server or nginx) and compute the time-to-compromise following McQueen et al. [18]. This information is added to the graph representation and additionally, the different solutions are combined. In other words, we prepare different graphs that to some degree contain e. g., Apache web server, while other graphs contain nginx or a combination of both.

[3]Bpmn2Mal prototype and mapping example for process from Ex. 1, visited 6/2021: https://github.com/dritter-hd/bpmn2mal

[4]NVD NIST, visited 6/2021: https://nvd.nist.gov/

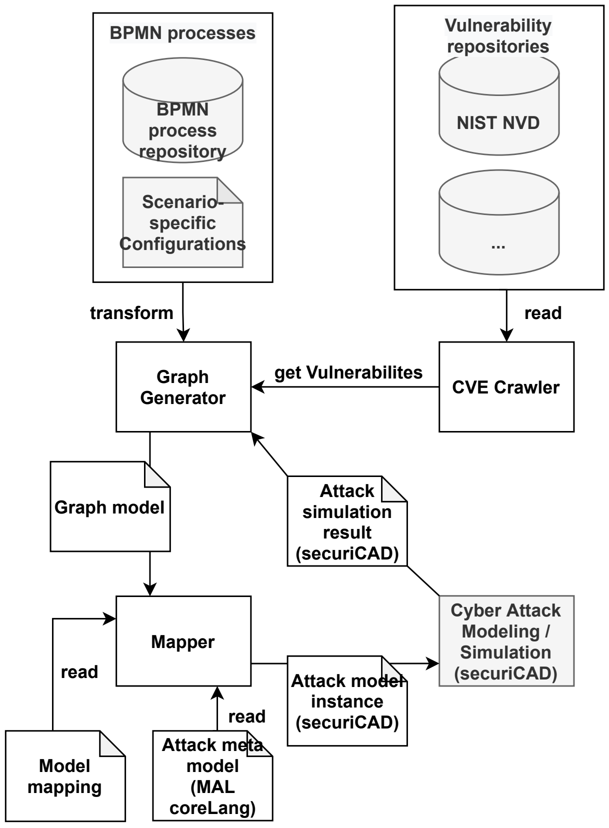
BPMN processes
BPMN process repository
Scenario-specific Configurations
Vulnerability repositories
NIST NVD
...
transform
read
Graph Generator
get Vulnerabilites
CVE Crawler
Graph model
Attack simulation result (securiCAD)
Cyber Attack Modeling / Simulation (securiCAD)
Mapper
read
Attack model instance (securiCAD)
read
Model mapping
Attack meta model (MAL coreLang)

Fig. 3. Prototype Architecture

These different graphs are then transformed into securiCAD models [5] that are instances of `coreLang` [15] following the mapping described in Sect. IV. This is extending previous work [7], which focused on automatically creating MAL languages, while here we are creating concrete instances of such languages. For each of the graphs, several simulations are performed giving the simulated attacker different attack surfaces (cf. Sect. IV-A). Then, the results of the different simulations should get aggregated and returned back to the graph representation to finally visualize the critical activities in the process model.

B. Motivating example (revisited): Italy invoicing

With Fig. 1, we introduced an example invoicing integration process. Facilitating the mapping presented in Sect. IV, we translate this integration process to the threat model presented in Fig. 4. To not clutter Fig. 1, it does not contain explicit data objects and related associations, which usually would be included. Accordingly, the respective `coreLang` representation does not contain explicit `Data` 🛢 either.

For greater clarity, Fig. 4 contains only a subset of the possible vulnerabilities 🛡 and exploits ☣. More concretely, just those that are needed to enable the attacker to write the data transmitted to *SdI-Production*, which is symbolized by the star on `Data` 🛢. The top of the threat model describes the communication between the *ERP-System* and the *Integration System*. In the lower left part, the different *Script Task* and

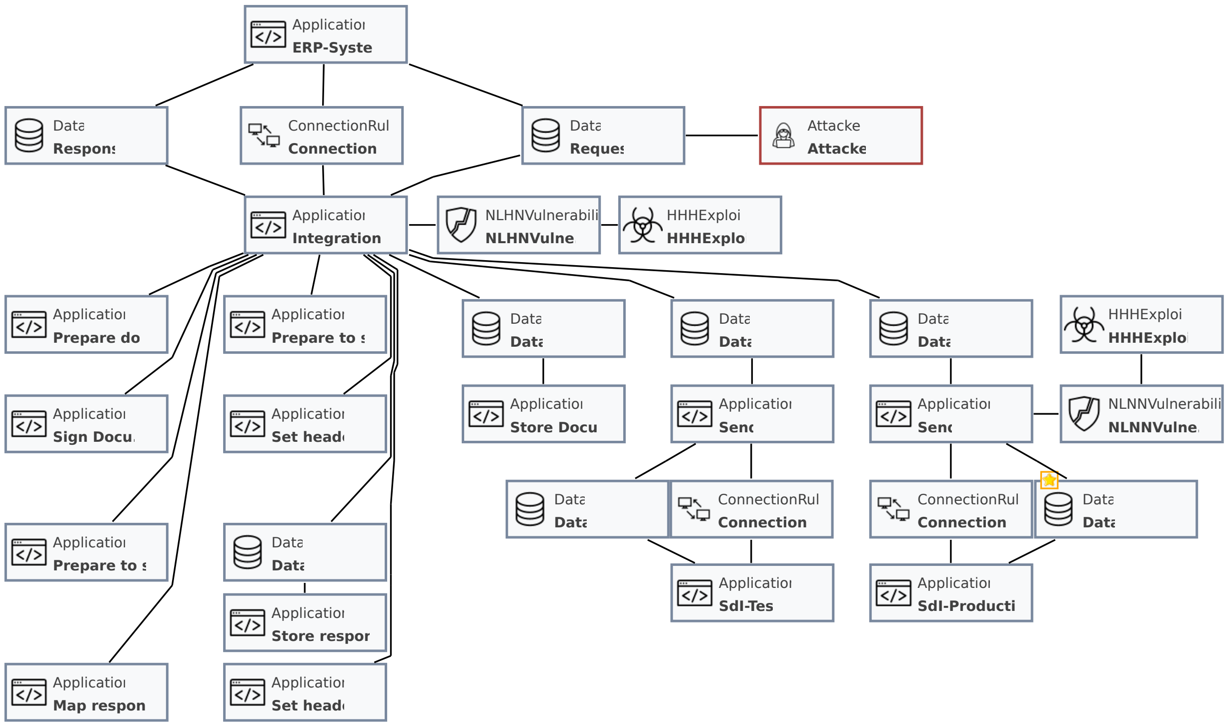

Fig. 4. Threat model representation of Fig. 1

Service Task take place, which mainly process the document. The lower right part describes the communication with the SdI for the test and the productive system.

Next, we discuss a penetration, where an attacker can perform a Man-In-The-Middle (MITM) attack on the request (i. e., `Data`) sent from *ERP-System* (i. e., `Application`) to *Integration System* (i. e., `Application`) via a `Connection`, e. g., because an older version of encryption is used. This enables the attacker to inject code into a component used by the process engine (CVE-2021-23337)[5]. As this vulnerability allows to execute arbitrary code, the attacker can alter the data sent to the *Service Task* communicating with SdI. If this *Service Task* is run by a miss-configured Apache Tomcat (CVE-2020-1938)[6], the attacker is again able to execute their desired code. Hence, it is possible to `write` on the related data sent to SdI and the attacker reached their goal.

Now, we are able to perform the attack simulation in securiCAD [5] to analyze what path the attacker takes through the process and determine possible countermeasures (cf. Fig. 5). The analysis shows that the actual design of the process can be considered as secure, as the described attacker achieves a success rate of 8% after 100 days of penetrating the system.

Nonetheless, some room for improvement was identified. To stop the attacker early, one can require that data transmissions are just accepted if the sender is properly authenticated, thus MITM would not be possible anymore. Another option, which would stop the attacker later, would be to patch the vulnerable applications. Then, these would not be exploitable anymore (at least for exploits related to these specific vulnerabilities).

[5]Lodash vulnerability used in Camunda, visited 6/2021: https://nvd.nist.gov/vuln/detail/CVE-2021-23337

[6]Apache Tomcat vulnerability, visited 6/2021: https://nvd.nist.gov/vuln/detail/CVE-2021-25329

C. Discussion

To reflect on our mapping decisions, we discussed them with an experienced threat modeler, who is also familiar with the concepts of `coreLang`. We explained the integration process in Fig. 1 and showcased the resulting threat model in Fig. 4. Overall, the expert agreed with our decisions. However, he raised some points that we discuss next.

In our mapping, we opted to map every *Script Task* to a single `Application`, because we assumed that every script could contain a vulnerability that might not be present in other scripts. Alternatively, one could group *Script Task* by the underlying technology, e. g., javascript, groovy, python, etc. Then, the focus would be more on the assumption that the vulnerability can be found in those technologies. Lastly, one could argue that there is no need for an explicit modeling of scripts since those are executed by the process engine. Discussing these three possibilities, we concluded that the first solution is the better one, as it contains information which scripts are corrupted, the data exchange between them is more obvious, and it is closer to the original process model.

Another point brought up was to split up the *Process / Participant* into several `Applications` that represent the

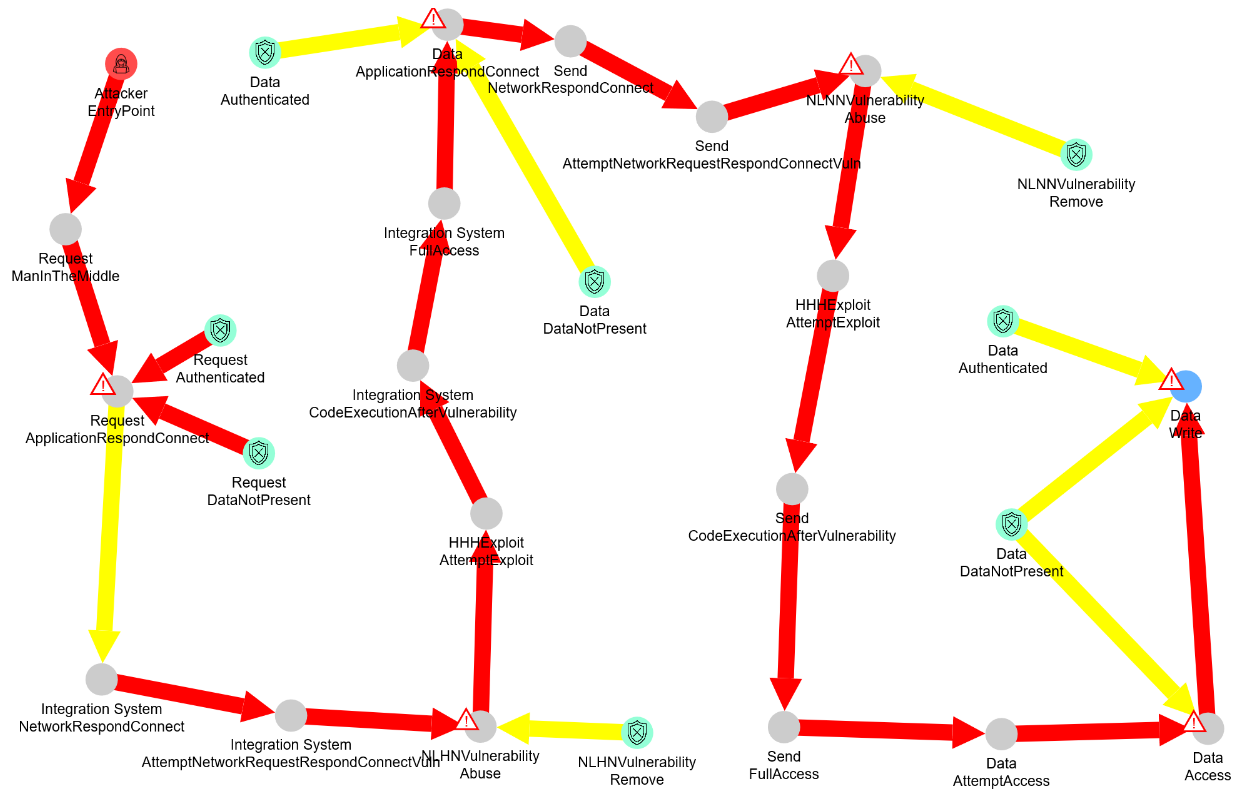

Fig. 5. Visualization of the attack path to reach `Data.Write` sent to `SdI-Production`

different functionalities. In our case, this would result in splitting the *Integration System* into two endpoints to receive the request and to send the response, and a component managing the process execution. However, we neglect this additional differentiation, as we do not see any added value and more elements would be presented in the final model.

Taking a closer look at the attack simulations presented before, we recognize that for a successful penetration of the process, the *Integration System* is crucial. To reach the other parts of the process, the easiest and shortest way usually facilitates this central component. This observation can be generalized for all integration processes that make use of a process engine. Thus, we can conclude that the security engineers should put emphasis on that component.

Another observation is related to the positioning of the attacker. In our example, the attacker can perform a MITM on the request. However, there are further attacks like eavesdropping, that could result in different outcomes of what is accessible to the attacker and what is not. Moreover, the attacker could be (theoretically) placed on each and every element of the process. However, the necessity of attacker placement is caused by the design of MAL itself, which could include an indicator, which assets and which attacks are the most likely attack surfaces.

This complexity is further increased by performing simulations for different configurations of the assets (i. e., a *Service Task* can either be executed by an Apache or an nginx http server). Each possible combination increases the number of needed simulations exponentially. Consequently, a solution is needed that reduces this number. One approach could be to restrict the possible configurations to realistic ones. For example, one can assume that within one organization (i. e., *Participant / Pool*), it is very likely that only one kind of a certain application for one purpose is deployed and only the versions might differ on a certain interval. Another possible assumption is to rely on usage statics of certain applications (e. g., http server) and to solely consider those that are used to a certain degree.

One aspect that can be improved in future is the fact that all our integration processes at hand only contain a subset of the possible BPMN elements. Especially, the integration processes are completely automatized and, hence, there is no interaction with humans expected. But the users of information systems are often used as attack surface to gain access to those systems. Moreover, human related security properties can differentiate significantly among organizations — and even in sub-entities of organizations. This cannot be codified in a static language such as `coreLang`. Therefore, an approach is needed that assess the security behavior in organizations (e. g., [6]) and includes this information into the performed attack simulations.

VI. Conclusion

To conclude our work, we revisit our objectives (i)–(iv). Objective (i) focused on the ability to assess the security of processes represented in BPMN. To achieve this objective, we created a mapping from BPMN to `coreLang`, which enables us to analyze the security of the respective process.

Objective (ii) suggested an automatic security assessment. We partly satisfy this objective in the current proposal. In our solution, the BPMN model is translated automatically to the format needed, however the simulations need to be manually started as securiCAD is missing a command interface at the moment. This would allow us to automatize this part as well. An alternative could be to use robotic process automation tools to automate the user interaction. Additionally, we cannot import the findings back to the original process model yet.

We have accomplished objectives (iii) and (iv). For objective (iii), we simply take the process model and enrich a graph representation of it with the needed security information, thus, no changes to the original model are necessary. For objective (iv), we use the information from the CVE databases to simulate different applications where we are not able to gain deeper knowledge of their characteristics.

A potential next step could be a user study, in which the attack simulation results are reviewed and categorized. It would be interesting to see how easy the findings are to fix. We assume that some findings would be; 1) a quick fix that can just be implemented without any further analysis, 2) doable but with some constraints and perhaps a more in-depth assessment, and 3) nearly impossible to do anything about without large efforts.

Further, we believe that an interesting study would be to analyze a set of old process implementations, review what security-related changes these processes have gone through and re-analyze the same processes in their new state. This, in order to see if our suggested approach did pick up issues that were fixed and also examine why other issues haven't been mitigated yet.

Acknowledgements We thank Rafael Lüdtke, Erik Henriksson, and Klas Engberg for implementation support, and Joar Jacobsson for valuable discussions on the BPMN to MAL mapping. This project has received funding from the Swedish Energy Agency.

References

[1] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. SecureBPMN: Modeling and enforcing access control requirements in business processes. In *SACMAT*, pages 123–126, 2012.

[2] Y. Cherdantseva and J. Hilton. A reference model of information assurance & security. In *ARES*, pages 546–555. IEEE, 2013.

[3] Y. Cherdantseva, J. Hilton, and O. Rana. Towards secureBPMN-aligning BPMN with the information assurance and security domain. In *BPMN*, pages 107–115. Springer, 2012.

[4] M. E. A. Chergui and S. M. Benslimane. Towards a BPMN security extension for the visualization of cyber security requirements. *IJTD*, 11(2):1–17, 2020.

[5] M. Ekstedt, P. Johnson, R. Lagerström, D. Gorton, J. Nydrén, and K. Shahzad. securiCAD by foreseeti: A CAD tool for enterprise cyber security management. In *EDOCW*, pages 152–155. IEEE, 2015.

[6] A. Georgiadou, S. Mouzakitis, K. Bounas, and D. Askounis. A cyber-security culture framework for assessing organization readiness. *Journal of Computer Information Systems*, 0(0):1–11, 2020.

[7] S. Hacks, A. Hacks, S. Katsikeas, B. Klaer, and R. Lagerström. Creating meta attack language instances using archimate: Applied to electric power and energy system cases. In *EDOC*, pages 88–97, 2019.

[8] S. Hacks and S. Katsikeas. Towards an ecosystem of domain specific languages for threat modeling. In M. La Rosa, S. Sadiq, and E. Teniente, editors, *Advanced Information Systems Engineering*, pages 3–18, Cham, 2021. Springer International Publishing.

[9] S. Hacks, S. Katsikeas, E. Ling, R. Lagerström, and M. Ekstedt. powerlang: a probabilistic attack simulation language for the power domain. *Energy Informatics*, 3(1), 2020.

[10] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[11] H. Holm, K. Shahzad, M. Buschle, and M. Ekstedt. P^{2}CySeMoL: Predictive, probabilistic cyber security modeling language. *TDSC*, 12(6):626–639, 2015.

[12] J. Jang-Jaccard and S. Nepal. A survey of emerging threats in cybersecurity. *J. Comput. Syst. Sci*, 80(5):973–993, 2014.

[13] T. Jeske, M. Würfels, F. Lennings, M.-A. Weber, and S. Stowasser. Achievements and opportunities of digitalization in productivity management. In *AHFE*, pages 17–24. Springer, 2020.

[14] P. Johnson, R. Lagerström, and M. Ekstedt. A meta language for threat modeling and attack simulations. In *ARES*, page 38. ACM, 2018.

[15] S. Katsikeas, S. Hacks, P. Johnson, M. Ekstedt, R. Lagerström, J. Jacobsson, M. Wällstedt, and P. Eliasson. An attack simulation language for the it domain. In H. Eades III and O. Gadyatskaya, editors, *GraMSec*, pages 67–86. Springer, 2020.

[16] C. L. Maines, D. Llewellyn-Jones, S. Tang, and B. Zhou. A cyber security ontology for BPMN-security extensions. In *CIT*, pages 1756–1763. IEEE, 2015.

[17] C. L. Maines, B. Zhou, S. Tang, and Q. Shi. Adding a third dimension to BPMN as a means of representing cyber security requirements. In *DeSE*, pages 105–110. IEEE, 2016.

[18] M. A. McQueen, W. F. Boyer, M. A. Flynn, and G. A. Beitel. Time-to-compromise model for cyber risk reduction estimation. In D. Gollmann, F. Massacci, and A. Yautsiukhin, editors, *Quality of Protection*, pages 49–64. Springer, 2006.

[19] P. H. Meland and E. A. Gjære. Representing threats in BPMN 2.0. In *ARES*, pages 542–550. IEEE, 2012.

[20] N. R. Moşteanu. Challenges for organizational structure and design as a result of digitalization and cybersecurity. In *CBER*, pages 278–286, 2020.

[21] I. Morikawa and Y. Yamaoka. Threat tree templates to ease difficulties in threat modeling. In *NBiS*, pages 673–678, Sep. 2011.

[22] J. Mülle, S. von Stackelberg, and K. Böhm. *A security language for BPMN process models*. KIT, Fakultät für Informatik, 2011.

[23] D. Ritter. Experiences with business process model and notation for modeling integration patterns. In J. Cabot and J. Rubin, editors, *ECMFA*, volume 8569, pages 254–266. Springer, 2014.

[24] D. Ritter and M. Holzleitner. Integration adapter modeling. In J. Zdravkovic, M. Kirikova, and P. Johannesson, editors, *CAiSE*, volume 9097, pages 468–482. Springer, 2015.

[25] D. Ritter, N. May, and S. Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Inf. Syst.*, 67:36–57, 2017.

[26] D. Ritter and J. Sosulski. Exception handling in message-based integration systems and modeling using BPMN. *Int. J. Cooperative Inf. Syst.*, 25(2):1650004:1–1650004:38, 2016.

[27] A. Rodríguez, E. Fernández-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE Trans. Inf. Syst.*, 90-D(4):745–752, 2007.

[28] M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and verifying security policies in business processes. In *BPMDS/EMMSAD*, pages 200–214. Springer, 2014.

[29] K. S. Sang and B. Zhou. BPMN security extensions for healthcare process. In *CIT*, pages 2340–2345. IEEE, 2015.

[30] SAP SE. Prepackaged cloud integration content. https://cloudintegration.hana.ondemand.com/, 2021.

[31] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren. Action design research. *MIS Quarterly*, 35(1):37–56, 2011.

[32] W. Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012.

[33] J. Venable, J. Pries-Heje, and R. Baskerville. Choosing a design science research methodology. In *ACIS*. University of Tasmania, 2017.

[34] S. Zareen, A. Akram, and S. Ahmad Khan. Security requirements engineering framework with BPMN 2.0. 2 extension model for development of information systems. *Applied Sciences*, 10(14):4981, 2020.