Maintaining Secure Business Processes in Light of Socio-Technical Systems' Evolution

Mattia Salnitri, Elda Paja, and Paolo Giorgini

University of Trento, Trento, Italy
{mattia.salnitri, elda.paja, paolo.giorgini}@unitn.it

Abstract. Today's systems are socio-technical: they consist of autonomous subsystems (social: humans and organizations, technical: software and hardware) that interact to get things done. Autonomy makes the design of secure socio-technical systems a challenging task that requires a consideration of both technical and socio-organizational concerns. Business process models offer an adequate level of abstraction to do so, for they capture how the various subsystems interact and allow the enforcement of security policies.
But, socio-technical systems are continuously evolving. Such an evolution inevitably affects the underlying business processes and the security policies they should comply with. Failing to maintain security policies enforced in business processes might result in security violations or law infringement.
To address this problem, we propose a framework for continuously and iteratively maintaining secure business processes by: (i) create business process models for the system-at-hand using the SecBPMN-ml language, (ii) capturing security policies using SecBPMN-Q, and (iii) verifying whether the first two are aligned (i.e., security policies are enforced in business processes) using a verification engine. We apply our framework on a case study about Air Traffic Control, and report on promising scalability results of our implementation.

1 Introduction

Software systems today operate in complex and dynamic environments, wherein they interact not only with other technical systems, but also with humans and organizations. This interplay of technical systems beyond the technical environment, involving social interactions too (with humans and organizations), constitutes what is known as a socio-technical system. Business processes are used to specify such interplay and the processes executed internally by the participants of such systems.

We find many examples of socio-technical systems in our everyday lives. Think of health care systems, e-commerce, social networks, but also of smart grid systems and Air Traffic Management (ATM). For example, in ATM socio-technical systems, technical systems such as tower control systems, meteo service providers and the luggage management system, execute business processes to interact with social actors such as control-tower operators, pilots, as well as companies that provide meteo information.

Security is a desirable feature in any system, especially so in socio-technical systems [10], due to their ever-increasing complexity and because social concerns and interactions among the various subsystems bring up many security issues (e.g., in an ATM

socio-technical system protecting the integrity of flight plans), the violation of which may have severe consequences (e.g., an unauthorized modifications of flight plans might threaten travelers' lifes).

Socio-technical systems are continuously evolving due to various possible changes: (i) system and subsystems changes, (ii) organizational and domain changes, and (iii) normative and regulatory changes. Any of these changes may affect the ability of the socio-technical system to meet its intended security requirements. However, security requirements themselves might evolve, and likewise there is a need to ensure that the socio-technical system satisfies the new security requirements too.

Therefore, in managing security in socio-technical systems, we need to consider such changes and update business processes and/or security policies, i.e. security requirements in terms of business process elements, accordingly. To maintain the alignment, that is, security policies are enforced in business processes, we need to verify updated business processes against security policies at each evolution step [1].

Existing approaches either maintain alignment by verifying business process properties other than security [3, 6] or do not take into account evolution [4, 22, 14]. In this work, we propose a framework to verify whether the business processes meet security policies in light of evolution by iteratively creating business process models for the system-at-hand, capturing security policies, and verifying whether the two are aligned.

Specifically, the contributions of this work are:

- A modeling language, namely SecBPMN2-ml, for modeling business processes;
- A modeling language, namely SecBPMN2-Q, for modeling procedural security policies;
- An implementation of our framework—using $DLV^{\mathcal{K}}$— as part of a CASE tool called STS-Tool;
- Results from a case study which show the effectiveness of our framework in identifying mis-alignments as well as promising scalability results.

The paper is structured as follows: Section 2 describes the motivating case study from the air traffic management domain. Section 3 presents the SecBPMN2-ml language, while Section 3 describes SecBPMN2-q language. Section 5 describes our implementation and Section 6 contains the evaluation results. Section 7 describes the relevant related work. Finally, Section 8 concludes.

2 The SWIM ATM socio-technical system, a case study

We consider here the System Wide Information Management (SWIM) ATM case study, which is a variant of the case study "*The emerging European Air Traffic Management systems*" of the project Aniketos [1]. The SWIM [8] will be introduced in ATM systems to support the collaboration and communications (including data exchange) of the various parties involved in the systems. Importantly, SWIM will allow the participation

[1] Note that the specifics of organizational, normative, and technological changes affecting business processes and security policies are out of the scope of this work, and they are considered as a black box.

of numerous external parties. These will access a common virtual pool of information, having data stored at different locations. However, SWIM will make ATM systems open (interconnecting parties from all over Europe), in which newcomers (even untrusted players) might also participate. This opens up many security issues, for the services offered by ATM through the SWIM vary from safety critical data to catering and lost luggage, and security breaches could happen at any point of communication.

Tackling the security problem in SWIM ATM, calls for an analysis of organizational, normative, and technological changes affecting ATM's business processes and security policies. Such an analysis is key to preserving the desired level of security.

Organizational changes might impact the execution of business processes. For example, if the pilots are not considered any more flights attendants, but are treated as independent participants of ATM, the business processes where they are involved will be updated because of reorganizations of the activities executed by the pilots. Such changes impact on security policies too, because all security policies where pilots are specified to be flight attendants, shall be updated.

Normative changes have a great impact on business processes. For example, a change in the privacy norms protecting sensitive data from disclosure, will impose a modification on the business processes where personal data are exchanged. Normative changes are strictly related to security policies too. For example, if a new privacy law imposes stricter norms on protecting the confidentiality of messages containing sensitive data, the security policies will need to be updated consequently.

Technological changes frequently affect business process. For example, an upgrade of flight systems with a functionality that automatically generates flight plans will modify the business processes where such plans are created. Such changes also impact on security policies. For example, an upgrade to the technology used to routing luggage may require availability of the customers' information every time the luggages are rerouted. This is transformed in a security policy that shall be checked to ensure the efficiency of the luggage system.

3 Modeling business processes with security choices

In this work, we propose SecBPMN2-ml, an extension of SecBPMN [24] that supports BPMN 2.0 and new security annotations (details provided below). Differently from other approaches, such as [4, 20], SecBPMN2-ml offers high expressiveness, and the supported security annotations make it applicable to a variety of domains.

Fig. 1 shows an example of a SecBPMN2-ml diagram. It represents two business processes delimited by a start event and an end event. Each business process is executed by a participant, namely *Tower control operator* and *Pilots* (that are further divided in *Captain* and *Co-pilot*), and contains at least one activity, e.g. *Analyze request take-off*. Communications between two participants are represented with message flows (thick dashed arrows), while the contents of the communications are represented by the *message* elements. For example, the execution of the activity *Send request take-off* creates a communication channel from the *Pilots* to the *Tower control operator* where the *Request* message is sent. The order of execution of activities is represented with control flow relations that links two SecBPMN2 elements and specifies that the source element

is executed before the target element. Gateways represent branches in the control flow, for example the diamond with an “X" symbol is an exclusive gateway. The gateway executed after the activity *Analyze request take-off* in Fig. 1 specifies that the control flow can take two different paths, based on the evaluation of the condition *Errors?*.

An execution of a SecBPMN2-ml model may result in different sequences of activities performed, depending on how the conditions in the gateways are evaluated. Such sequences are called paths, defined in Def. 1. The definition is adapted from path definition in graph theory [29]. For example, in the SecBPMN2-ml model represented in Fig. 1, there are 4 paths: 1 for the upper process and 3 for the lower process. For example, the path of the lower process [*Start*,*Analyze request take-off*,*Send rejection*, *End*].

Definition 1 *Given bp a set containing all elements of a SecBPMN2-ml model, we define path(a,b), with a,b* $\in$ *bp as a sequence of elements of bp, that starts at a and ends at b, where consecutive elements in the sequence are connected by a control flow in bp, and no element is repeated.*

SecBPMN-ml allowed specifying eight security annotations, namely: *accountability*, *auditability*, *authenticity*, *availability*, *confidentiality*, *integrity*, *non-repudiation* and *privacy*. These annotations represent security concepts defined in the Reference Model for Information Assurance and Security (RMIAS) [5]. The security annotations are graphically represented with a solid orange circle, with a black icon, that changes depending on the type of security annotation. In SecBPMN2-ml, we have defined the semantics of all the security annotations when linked to new elements introduced in BPMN 2.0 (see details in the technical report [26]), and we have added 3 new security annotations, namely *separation of duties*, *bind of duties* and *non-delegation*. The added security annotations originate from our collaboration with security experts and practitioners, and from our experience of applying the language to numerous case studies. For each security annotation it is possible to specify attributes, called security properties, for specifying further details about the security annotations. For example, *enforcedBy* can be used to specify configurations, software and hardware, that will be used to enforce the security annotation in the implementation of the business process.

Separation of duties is a security principle used to formulate multi-person control policies, requiring that two or more different people be responsible for the completion of a task or set of related tasks [27]. If the set of people changes during the execution of the system, we have dynamic separation of duties, otherwise, we have static separation of duties. We have created the *Separation of duties* security annotation, in which it is possible to set two security properties: (i) *enforcedBy*—explained above; (ii)*dynamic*—a Boolean value set to true when dynamic separation of duties is required, false otherwise. For example, in Fig. 1 a separation of duties security annotation is linked to *Captain* and *Co-pilots* because a single person cannot execute at the same time the activities of captain and co-pilots. In this case, *dynamic* security property shall be set to true, because, a person, in different occasions, can be either the *Captain* or the *Co-pilots*.

Bind of duties requires the same person to be responsible for the completion of a set of related tasks [28]. Similarly to separation of duties, we have static and dynamic bind of duties. Such security concept is represented by *Bind of duties* security annotations in

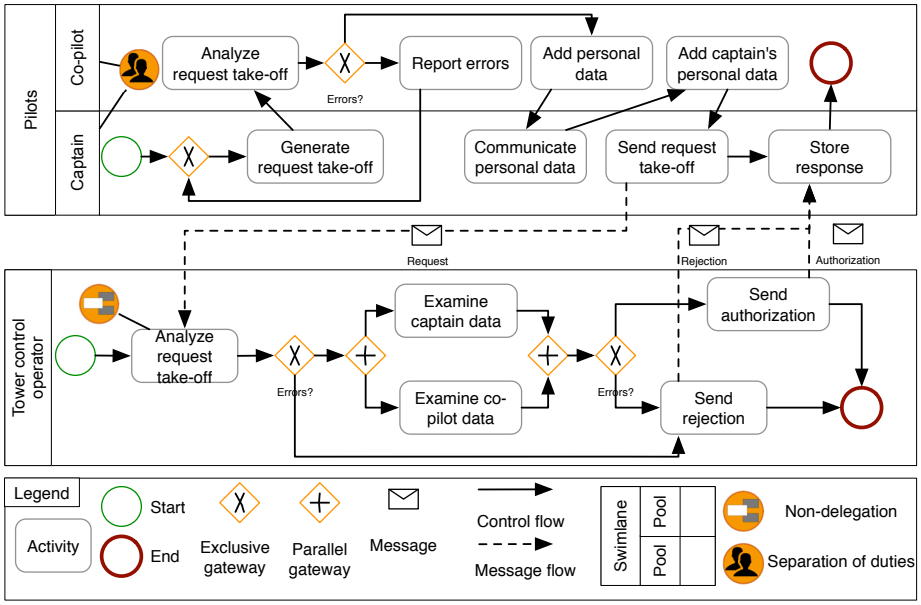


Fig. 1: Example of a SecBPMN2-ml business process.

which it is possible to specify the same security properties of separation of duties. For example, in the business process in Fig. 1, the *Tower control operator* and *Co-pilots* would have been connected to a bind of duty security annotation if their activity was to be executed by the same person.

Non-delegation requires that a set of actions shall be executed only by the users assigned. It is represented by *Non-delegation* security annotation, in which it is possible to specify two security properties: (i) *enforcedBy*—explained above; (ii) *allowedDelegations*, a number that specifies how many times it is possible to delegate the task to other participants. For example, in the business process in Fig. 1, since *Non-delegation* is linked to *Analyze request take-off*, the activity will be executed by the *Tower control operator* alone, and will not be delegated to other participants.

4 Modeling and verifying procedural security policies

In this paper, we define SecBPMN2-Q, a modeling language expressly thought for the specification of procedural security policies. It builds on top of SecBPMN-Q [24], a query language (part of SecBPMN) that differently from other approaches [3, 6] is aimed at verifying security policies over business process models, and thus is suited to our needs. SecBPMN2-Q extends SecBPMN-Q with BPMN 2.0 and the vast range of security annotations supported by SecBPMN2-ml.

SecBPMN2-Q contains three relations that are not defined in BPMN 2.0: *negative flows*, *path* and *negative path*. All of them connect two SecBPMN2-ml elements among

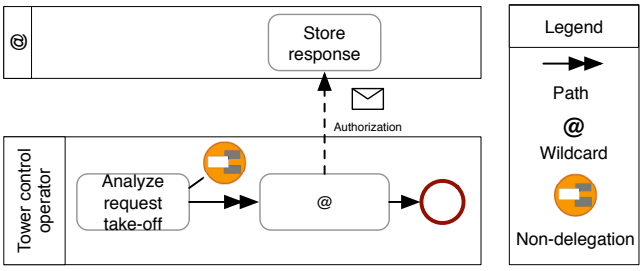


Fig. 2: Examples of a SecBPMN2-Q security policy.

activities, events or gateways. **Negative flow relation** matches all business processes in which the target element is never executed right after the source element. **Path relation** matches all business process in which the target element is executed after the source element. **Negative path relation** is the negation of path relation, i.e., it matches all the business processes in which the target element is never executed after the source element.

SecBPMN2-ml assigns a specific semantics to the label "@", which matches all elements of the same type irrespective of the label.

Figure 2 shows an example of a SecBPMN2-Q security policy that contains *Tower control operator* and @ swimlanes, *Analyze request take off* activity, a path relation between the latter activity and the activity @, etc.

The verification of the enforcement of security policies in business processes consists in checking if all the security annotations in security policies are enforced in the business process, and if there exists at least one path in the business process that matches the path specified in the security policies.

Definition 2 specifies when a SecBPMN2-Q security policy is considered aligned with a SecBPMN2-ml business process. The set *sps* obtained from the function *substituteWildcards(sp,bp)* contains all the possible instantiation of the security policy where elements labeled with @ wildcard are substituted with elements, of the same type, of the SecBPMN2-ml model. The alignment is verified if there exists one instance of the security policy where (i) all elements and relations in the security policy instance, except for path, negative path and negative control flow relations, are contained also in the business process; (ii) all security annotations in the security policy instance are *enforced* (see def. 3) by *comp* (i.e., compatible) security annotations in the business process. The *comp* predicate is true if: (a) the security annotations are the same type, and (b) the activities associated to the security annotations have the same name. Furthermore, (iii) at least one path matches all the path specified in the security policies instance, (iv) no paths matches the negative path in the security policy instance; and finally (v) no control flow matches the negative control flow in the security policy instance.

Definition 2 *Given bp and sp, two sets containing respectively all elements in a SecBPMN2-ml model and in a SecBPMN2-Q security policy ,*
bp_{Ann} *and* sp_{Ann} *the set of security annotations respectively in bp and sp,*
bp_{Path} *the set of all paths in bp,*

sps = substituteWildcards(sp,bp),
sp is aligned with bp iff $\exists spi \in sps$
where $spi_{pr}, spi_{npr}, spi_{nfr}$ *are respectively the set of all paths, negative path and negative flow relations in spi* $\wedge$
(i) $spi \setminus bp$ *contains only path, negative path and negative control flow relations* $\wedge$
(ii) $\forall\ sa_{sp} \in sp_{Ann} \wedge \forall\ sa_{bp} \in bp_{Ann}$ *comp(*sa_{bp}, sa_{sp}*)* $\rightarrow$ *enforced(*(sa_{bp}, sa_{sp}) $\wedge$
(iii) $\exists$ *p(*t_j, t_m*)* $\in$ bp_{Path} *(where p(*t_j, t_m*) =* $[t_j, .., t_k, ..t_l, ..t_m]$ *) s.t.* $\forall\ pr(a, b) \in spi_{pr} t_k = a \wedge t_l = b$, *with j≤k<l≤m* $\wedge$
(iv) $\nexists$ *p(*t_j, t_m*)* $\in$ bp_{Path} *(where p(*t_j, t_m*) =* $[t_j, .., t_k, ..t_l, ..t_m]$ *) s.t.* $\forall\ npr(a, b) \in spi_{npr} t_k = a \wedge t_l = b$, *with j≤k<l≤m* $\wedge$
(v) $\nexists$ *p(*t_j, t_m*)* $\in$ bp_{Path} *(where p(*t_j, t_m*) =* $[t_j, .., t_k, ..t_{k+1}, ..t_m]$ *) s.t.* $\forall\ nfr(a, b) \in spi_{nfr} t_k = a \wedge t_{k+1} = b$ *with j≤k≤m* $\wedge j \neq m$

For example, the business process in Fig. 1 enforces the security policy in Fig. 2 because in the security policy instance where the @ activity is substituted with *Send authorization*: (i) the security policy instance is a subset of the elements of the business process, except for path, negative path and negative flow relations; (ii) non-delegation security annotation is enforced by *Analize request take-off* executed by *Tower control operator*; (iii) there exists a path where an *Authorization* message is handled by *Store response* and it is sent by *Tower control operator* after the execution of *Analize request take-off*. Points (iv) and (v) does not apply because there are no negative path and control flow relations in the security policy instance.

Definition 3 specifies when a security annotation in a security policy is enforced by a *compatible* security annotation in a business process: when all security properties of the latter define stricter security properties of the former. For example, *enforcedBy* security property in a business process is “stricter" that the one in a security policy if the set the former is a sub set of the set in the latter.

Points (i)-(xi) in Definition 3 specify the criteria for each type of security annotation. The rationale behind points (i)-(viii) is described in [24]. Points (ix) and (x) specify that the value of *dynamic* security property, of respectively separation and bind of duties, must be the same in both the security annotations. Point (xi) specifies that the number of allowed re-delegations shall be lower in the business process than in the security policy. The function *type* returns the type of the security annotation received in input.

Definition 3 *Given two security annotations* sa_{bp}, sa_{sp}
We define enforced(sa_{bp}, sa_{sp}*) as a predicate that is true iff enforcedBy(*sa_{bp}*)* $\supseteq$ *enforcedBy(*sa_{sp}*) and*
(i) *type(*sa_{bp}*) = accountability* $\rightarrow$ *monitored(*sa_{bp}*)* $\subseteq$ *monitored(*sa_{sp}*),*
(ii) *type(*sa_{bp}*) = auditability* $\rightarrow$ *frequency(*sa_{bp}*)* $\leq$ *frequency(*sa_{sp}*),*
(iii) *type(*sa_{bp}*) = authenticity* $\rightarrow$ *(identification(*sa_{bp}*)* $\rightarrow$ *identification(*sa_{sp}*) and authorization(*sa_{bp}*)* $\rightarrow$ *authorization(*sa_{sp}*) and trustValue(*sa_{bp}*)* $\leq$ *trustValue(*sa_{sp}*)),*
(iv) *type(*sa_{bp}*) = availability* $\rightarrow$ *value(*sa_{bp}*)* $\leq$ *value(*sa_{sp}*)*
(v) *type(*sa_{bp}*) = confidentiality* $\rightarrow$ *(writers(*sa_{bp}*)* $\subseteq$ *writers(*sa_{sp}*) and readers(*sa_{bp}*)* $\subseteq$ *readers(*sa_{sp}*)),*
(vi) *type(*sa_{bp}*) = integrity* $\rightarrow$ *(personnel(*sa_{bp}*)* $\rightarrow$ *personnel(*sa_{sp}*) and hardware(*sa_{bp}*)* $\rightarrow$ *hardware(*sa_{sp}*) and software(*sa_{bp}*)* $\rightarrow$ *software(*sa_{sp}*)),*

(vii) *type(sa_{bp}) = non-repudiation → (execution(sa_{bp}) ↔ execution(sa_{sp})),*
(viii) *type(sa_{bp}) = privacy →(sensitiveInfo(sa_{bp}) ⊆ sensitiveInfo(sa_{sp})),*
(ix) *type(sa_{bp}) = separation of duty → (dynamic(sa_{bp}) ↔ dynamic(sa_{sp})),*
(x) *type(sa_{bp}) = bind of duties → (dynamic(sa_{bp}) ↔ dynamic(sa_{sp})),*
(xi) *type(sa_{bp}) = non-delegation → allowedDelegations(sa_{bp}) ≤ allowedDelegations(sa_{sp})*

For example, the verification of the enforcement of the security annotations in the security policy in Fig. 2 against SecBPMN2-ml model in Fig. 1 is positive when: (a) both *Separation of duties* security annotations have their *dynamic* security property set to true (point (ix)); (b) *allowedDelegations* security property in the security policy is greater or equal of the same security property in the business process (point (ix)) .

5 Automated verification

When dealing with real world scenarios business processes can be considerably large, with hundreds of elements and tens of participants, therefore automating the verification of security policies against business processes is essential.

We built a verification engine on top of a planner for checking business processes with security choices against security policies. Planners, e.g.$DLV^{\mathcal{K}}$ [7] or PDDL4J [17], are software engines created to generate strategies that respect a set of constraints and achieve a set of goals.

We used the set of constraints so that the possible strategies generated reflect the possible executions of the business process. For example, the business process in Fig. 1 executed by *Co-pilot* is transformed in a set of constraints for which the generated strategies are: (i) *Generate request take-off*, *Analyze request take-off*, *Report errors*, etc.; (ii) *generate request take-off*, *Analyze request take-off*, *Add personal data*, etc.

Security policies are transformed to goals, i.e., in predicates that must be true once the strategy is executed. For example, the security policy in Fig. 2 is transformed into a goal that specifies that: (i) the message *Authorization* is sent from any activity executed by *Tower control operator*; and (ii) *Analyze request take-off* is executed before the message is sent.

Based on this, if it is possible to find a strategy (i.e., an execution of the business process) that reaches the goals, then the business process satisfies the security policy.

We use $\mathcal{K}$ [7] as a planning language to define the set of constraints and the goals. $\mathcal{K}$ is supported by $DLV^{\mathcal{K}}$ [7], a software engine that generates strategies starting from a $\mathcal{K}$ specification. Listing 1 shows an example of a $\mathcal{K}$ representation of a SecBPMN2 procedural model. In $\mathcal{K}$ strings that start with an upper case character are considered variables, while strings that start with a lower case character are constants. The former represent any activity and the latter represent specific activities. Lines 1-3 define the message and the control flow fluents. Lines 4-8 represent part of the procedural model in Fig. 1. In particular, Line 5 represents a control flow, while Line 6 specifies a message flow. Lines 7-8 specify the security annotation linked to the activity *Analyze request take-off*. Line 7 specifies the existence of the security annotations. Line 8 specifies the value of security property allowedDelegations.

```
1 fluents:
2   controlFlow(T1,T2) requires activity(T1), activity(T1).
3   messageFlow(T1,T2,M) requires activity(T1), activity(T1), message(M).
4 initially:
5   controlFlow(add captain's personal data, send request take-off).
6   messageFlow(send request take-off, analyze request take-off, request).
7   nonDel (analyze request take-off).
8   nonDel_allowedDel(analyze request take-off, 0).
```

Listing 1: A $\mathcal{K}$ representation of part of the process in Fig. 1

Listing 2 shows the $\mathcal{K}$ goal generated from the procedural security policy in Fig. 2. Line 1 specifies that a $\mathcal{K}$ goal is defined. Lines 2-3 contain the predicate *executed* that indicates that the activity specified in the first parameter must be executed by the participant indicated by the second parameter. Line 4 specifies that the strategy will contain a message sent by any activity to *store response* and transmitting the message *authorization*. Lines 5-6 specify that the activity *Analyze request take-off* shall be annotated with a non-delegation security annotation (Line 5) and that the value of the security property allowed delegation should be lower or equal to 2 (Line 6). Line 7 instructs the planner to find a strategy where *path1* is executed. *path1* is defined as a set of constraints (Lines 9-10), where *path1_1* becomes true after *analyze request take-off* is executed and *path1* becomes true if the same activity that sent the message defined in Line 4 is executed and *path1_1* is true. Line 8 specifies how many actions the generated strategy shall contain in maximum, we set this as the number of all SecBPMN2 elements in the business process. This parameter does not influence the generated strategies, if it is high enough, i.e., as high as the longest possible execution, which in the case of this paper is the number of elements in the business process.

```
1  goal:
2    executed(analyze request take off, tower control operator),
3    executed(store response, A),
4    sent(X, store response, authorization),
5    nonDel (analyze request take-off).
6    nonDel_allowedDel(analyze request take-off, 2).
7    path1
8    ? (10).
9  caused path1_1 after exec(analyze request take off).
10 caused path1 after exec(X), path1_1.
```

Listing 2: Example of a $\mathcal{K}$ goal generated from Fig. 2

For the sake of space, we do not include the transformation rules between SecBPMN2 and $\mathcal{K}$. Documentation on such rules can be found at [26].

If $DLV^{\mathcal{K}}$ generates at least a strategy that respects the constraints defined in Listing 1, which achieves the goal defined in Listing 2, then the business process used to generate the constrains is verified against the security policy used to generate the goal.

6 Evaluation

The purpose of the evaluation is twofold: (i) show the effectiveness of the framework in verifying security policies against business processes through a real case study, and (ii) show that the framework scales well with large SecBPMN2-ml models.

6.1 Findings from the case study

We evaluated the framework using the SWIM ATM case study. We analyzed a set of requirement documents produced by experts of the EU FP7 Project Aniketos [2], documents that specify SWIM ATM business processes. We modeled four large business processes: management of external services, usage of external services, negotiation of the flight plan, and landing. We modeled 60 security policies and we verified the alignment against the said business processes. The first verification highlighted errors due to misunderstandings of the documentation. We modified the processes and the policies to fix the problems and the verification of the alignment, this time, was positive. For example, we modified a policy which initially required a separation of duties between all roles played inside a plane: such policy was to general, indeed applies to all sub-roles of flight attendants which, frequently, are played by the same employee. We modified the policy by specifying which roles in the plane requires a separation of duties, i.e. *captain*, *co-pilots* and *flight attendants*.

After this experience, SecBPMN2-ml was found to be quite expressive in specifying all the characteristics of the business processes and security policies. In a previous research work [25], we evaluated the expressiveness of SecBPMN using the SWIM ATM case study. Since we used the same case study, we compared the results of the two evaluations. SecBPMN2 is more expressive, we were able to specify more characteristics of the business processes and security policies. But SecBPMN2 is more complex and, therefore more difficult to master for inexperienced modelers. The complexity was inherited from BPMN 2.0 that is, never the less, widely adopted. Anyhow, from our experience, a superficial knowledge of SecBPMN2 permits inexperienced modelers to specify business processes that are refined later, once the modelers gain a more complete knowledge of the modeling language.

Due to a change of the privacy law normative and organizational parts of the SWIM ATM system were updated. Therefore, business processes and security policies had to be updated, and the alignment verified.

The change in the privacy law imposes restrictions over the transmission of sensitive information, stating that sensitive information shall not be available to unauthorized users when transmitted, even in the internal network of the system. We updated both the business processes and the security policies in order to be compliant with the new regulation and to reflect what the law specifies. After such modifications the verification of the alignment was negative. The updated security policy was not enforced in the business process about the usage of external services: sensitive information about the pilots, contained in the flight plans, was transmitted before the usage of the meteo external service, to the tower control. We, therefore, modified the business process, by linking a confidentiality security annotation to the communication between the tower control and the plane, and ran again the verification and, this time, they were aligned.

6.2 Scalability evaluation

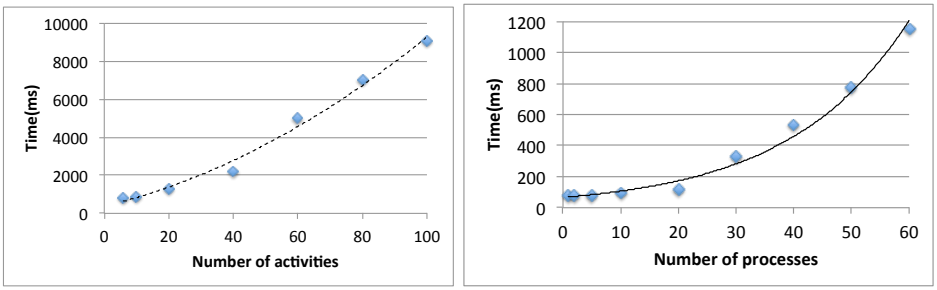
We performed a scalability study to evaluate how well the verification software engine would scale up with large models and to find which factors of the problem, i.e. elements

[2] `http://www.aniketos.eu`

of SecBPMN2-ml, influence its performances. We used time as a metric to test the scalability of the software.

Design of the experiment. We identified 8 factors that increase the complexity of the problem, namely the number of activities, participants, data objects, events, processes, paths, message flows, and security annotations. We executed each test five times, we excluded the highest and lowest time of execution and we calculate the average of the three remaining values.

Results. The tests were executed with a Mac Book Pro early 2011, with 8 GB of memory, Processor Intel Core i5 2,3 GHz, powered by OS X Yosemite 10.10.2. The results of the tests indicate that the impact of the factors grows linearly except for the number of activities, that is polynomial, and for the number of processes, that is exponential. Fig. 3a shows the results of the tests about the number of activities, while Fig. 3b shows the results for the number of the processes. The results indicate that the framework scales well: the exponential growth related to the number of processes does not influence significantly the overall scalability because, from our experience, even in very big SecBPMN2-ml models, the number of processes is frequently below 10. For further information on the results and the $DLV^{\mathcal{K}}$ files used for the tests, please refer to [26].



(a) Results tests on number of activities.

(b) Results tests on number of processes.

Fig. 3: Results of scalability tests.

7 Related work

In the years past, several approaches have been proposed to address the verification of requirements in business processes [4, 20, 30, 22, 14]. However, as far as our knowledge goes, there are no approaches that cover the overall security requirements engineering and verification process proposed in this paper.

In the following, we describe the most known approaches, while highlighting the differences with our proposal.

Modeling BPMN with security concepts for procedural specification. Many graphical modeling languages extending BPMN [16] with security aspects have been pro-

posed. SecureBPMN [4], proposed by Brucker et al., captures security and compliance requirements. Other extensions of BPMN also rely on security annotated business process modelling [20, 30, 22, 14] similarly to SecBPMN2. However, differently from existing approaches, ours allows the definition of custom security policies. Instead, existing approaches employ software engines that check a fixed set of hard coded security policies. Examples of such engines include [30, 23, 21].

Modeling security policies. Graphical query languages have been proposed to check if a process satisfies a query, which can be interpreted as a policy. For instance, BP-QL (Business Process - Query Language) [3] and BPQL (Business Process Query Language) [6] allow to graphically define queries and check which business processes satisfy the queries. These two query languages are not based on BPMN2, which makes their applicability and, most importantly, their learning process slower than that of, for example, SecBPMN2-Q that is based on BPMN 2.0.

Verification of security policies. Some approaches build on logic languages (e.g., first-order, temporal, etc.) for determining compliance. These works are characterized by high expressiveness, but poor usability, for they require a substantial effort for formalizing business processes and security policies.

Sadiq et al. [21] propose to use a Formal Contract Language (FCL) to express normative specifications. Their approach includes a modeling language to visualize business processes as well as normative constraints. They also define a compliance distance, which denotes the extent to which the process model has to be changed to become compliant with the declared constraints. The limitation of this approach is the complexity of the language, despite the provision of a tool to graphical represent normative requirements and business processes.

Liu et al. [12] describe how to check the compliance between a set of formally expressed regulatory requirements and business processes. The approach is accompanied by a software that verifies the business process against these compliance rules through the use of model-checking technologies. Their approach uses Business Process Execution Language (BPEL) [15] instead of BPMN 2.0, and it is not focused on security, but rather on regulatory compliance.

Ghose and Koliadis [11] enrich BPMN with annotations, and calculate how much a business process deviates from another business process. Differently from our approach, theirs focuses only on the structural difference between processes with no consideration of security requirements.

Other works [2, 19] use extensions of Petri nets to define business processes with security choices of stakeholders. Petri nets modeling language is simple and easy to use, but it does not include all the graphical constructs of BPMN. This influences negatively the understandability of models about medium-size or large business processes, limiting the applicability to only small-size business processes.

Planning languages We built the software engine, which verifies security policies against business processes, on top of $\mathcal{K}$. Other planning languages have been proposed. Stanford Research Institute Problem Solver (STRIPS) [9] is one of the first planning language. $\mathcal{K}$ is based on its syntax: it extends STRIPS with, for example, the possibility to put more than one executability conditions on actions. Action Definition Language

(ADL) [18] and Planning Domain Definition Language (PDDL) [13] are planning languages created on top of STRIPS. We chose $\mathcal{K}$ instead of STRIPS, PDDL or ADL, because of its expressiveness and because it generates strategies from an incomplete description of the domain., i.e., when the description of the domain does not cover all the possible situations. The second case is frequent in socio-technical systems systems where the business processes are not well defined and only a list of possible actions in known.

8 Conclusions and future work

We have proposed a framework for maintaining the alignment between business processes and security policies throughout the evolution of socio-technical systems. The framework includes: (i) SecBPMN2-ml, a modeling language for business processes with security aspects; (ii) SecBPMN2-Q, a modeling language for procedural security policies; (iii) a software engine for the verification of security policies against business processes. We evaluated the modeling languages with an ATM case study.

The framework suffers from the these limitations: (i) inability to define custom security policies; (ii) the SecBPMN2-ml and SecBPMN2-Q are built on top of BPMN 2.0, and they force users, which have a background on other modeling languages for business processes, to learn a new, complex, language.

Future work consider: (i) a complete empirical evaluation of the framework with practitioners; (ii) a quantitative evaluation of the software engine to test its scalability; (iii) an extension of the modeling languages to allow to define custom security annotations; and last, but not least (iv) involve domain experts in the process for a more interactive reflection of business process and/or security policy changes, and to establish an interactive decision-making process.

Acknowledgement

This research was partially supported by the ERC advanced grant 267856, 'Lucretius: Foundations for Software Evolution', `www.lucretius.eu`.

References

1. Aniketos Website. Last visited March '15. http://aniketos.eu.
2. V. Atluri and W. Huang. An Extended Petri Net Model for Supporting Workflows in a Multilevel Secure Environment. In *Database Security X '96*, pages 199–216.
3. C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Information Systems*, 33(6):477–507, 2008.
4. A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In *SACMAT '12*, pages 123–126.
5. Y. Cherdantseva and J. Hilton. A Reference Model of Information Assurance and Security. In *Proc. of ARES*, pages 546–555, 2013.

6. D. Deutch and T. Milo. Querying Structural and Behavioral Properties of Business Processes. In *DPL '07*, pages 169–185.
7. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete knowledge. In *CL '00*, pages 807–821.
8. Eurocontrol. System wide information management (swim), April 2013.
9. R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. IJCAI'71*, pages 608–620.
10. Tony Flick and Justin Morehouse. *Securing the smart grid: next generation power grid security*. Elsevier, 2010.
11. A. Ghose and G. Koliadis. Auditing Business Process Compliance. In *ISOC '07*, pages 169–180.
12. Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM System Journal*, 46(2):335–361, 2007.
13. D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.
14. M. Menzel, I. Thomas, and C. Meinel. Security Requirements Specification in Service-Oriented Business Process Management. In *ARES '09*, pages 41–48.
15. OASIS. Web Services Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, Apr 2007.
16. OMG. BPMN 2.0, Jan 2011.
17. PDDL4J website. Last visited March '15. http://sourceforge.net/projects/pdd4j/.
18. E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR '89*, pages 324–332.
19. J. L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In *ICATPN '96*, pages 400–419.
20. A. Rodríguez, E. Fernández-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE Trans. on Information and Systems*, pages 745–752, 2007.
21. S. Sadiq, G. Governatori, and K. Namiri. Modeling Control Objectives for Business Process Compliance. In *BPM '07*, pages 149–164.
22. M. Saleem, J. Jaafar, and M. Hassan. A Domain- Specific Language for Modelling Security Objectives in a Business Process Models of SOA Applications. *AISS '12*, 4(1):353–362.
23. M. Salnitri, F. Dalpiaz, and P. Giorgini. Aligning Service-Oriented Architectures with Security Requirements. In *OTM '12*, pages 232–249.
24. M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and Verifying Security Policies in Business Processes. *BPMDS '14*, pages 200–214.
25. M. Salnitri and P. Giorgini. Modeling and Verification of ATM Security Policies with SecBPMN. *In Proc. of SHPCS*, 2014.
26. SecBPMN Website. SecBPMN website, last visited March 2015. http://www.secbpmn.disi.unitn.it.
27. R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. In *Proc. of CSFW*, pages 183–194, 1997.
28. J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC - a workflow security model incorporating controlled overriding of constraints. *ijcis*, 12, 2003.
29. R. J. Wilson. *Introduction to Graph Theory*. John Wiley & Sons, Inc., 1986.
30. C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *JSA*, 55(4):211 – 223, 2009.