2016 IEEE 24th International Requirements Engineering Conference Workshops

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

Mattia Salnitri
DISI, University of Trento
Trento, Italy
Email: mattia.salnitri@unitn.it

Elda Paja
DISI, University of Trento
Trento, Italy
Email: elda.paja@unitn.it

Paolo Giorgini
DISI, University of Trento
Trento, Italy
Email: paolo.giorgini@unitn.it

***Abstract*—Today's systems are socio-technical, they are composed of social (humans and organizations) and technical components that interact with one another to achieve objectives they cannot achieve on their own. Security is a central issue in socio-technical systems and cannot be tackled through technical mechanisms alone. Instead, it requires enforcing security policies over the procedures that specify how components of these systems operate and interact (i.e., business processes). The continuous evolution of socio-technical systems, to adapt to external changes, may result in business processes that do not enforce security. Thus, it is important to preserve security through a constant update of business processes and/or security policies, to avoid security issues that may result in loss of reputation or monetary sanctions. To this end, in this paper we propose a framework to assist security engineers in maintaining secure business processes during socio-technical systems evolution. The framework is composed of: (i) SecBPMN2-ml, a modeling language for business processes; (ii) SecBPMN2-Q, a modeling language for security policies; and (iii) a software engine that verifies if security policies are enforced in business processes. The framework is applied to a case from the air traffic management domain.**

I. Introduction

Today's companies invest remarkably in reaching and maintaining a level of security that is either demanded by their stakeholders or dictated by security standards or regulations they should comply with [1]. In particular, the verification of security requirements in business processes, i.e., procedures executed to achieve companies' business objectives, is a central issue [11]. Such verification is a time consuming and error-prone task, because of the dimensions and of the complexity of business processes. It is, however, a necessary task, to prove that a given company (in particular its processes) is compliant with the security requirements specified by the stakeholders or by security standards and regulations.

This is a challenging task since today's systems can no longer be treated as monolithic entities, rather they are designed and developed together with their underlying environment, involving people, organizations, and technical components [15].

Such systems are continuously evolving, either because new technologies, new laws or standards are introduced, or because stakeholders' requirements change. Whatever the trigger, these changes invariably affect the ability of the socio-technical system to meet its intended security requirements. What is worth emphasizing is that the changes can hardly be planned in advance.

Therefore, in managing security in ever changing socio-technical systems, the best we can do is verifying that security is maintained, by keeping security policies satisfied in business processes.

Existing approaches either verify properties other than security [4], [7], or do not consider evolution [5], [19], [13]. In this work, we propose a framework to verify whether the business processes meet security policies by iteratively creating business process models for the system at hand, capturing security policies, and verifying if the latter are satisfied by the former. Specifically, the contributions of this work are:

- a modeling language, namely Secure BPMN 2.0 (SecBPMN2), for modeling business processes and specifying procedural security policies;
- a formal specification of the modeling language and of the verification of business processes against security policies;
- an implementation of the verification using $DLV^{\mathcal{K}}$ [8];
- a case study to evaluate the effectiveness of the framework in handling violations of security policies, as well as promising scalability results of our implementation.

The paper is structured as follows: Section II describes the overall framework. Section III presents the SecBPMN2 modeling language, while Section IV describes the verification framework. Section V introduces our implementation and Section VI presents the evaluation results. Section VII describes the relevant related work. Finally, Section VIII describes the conclusions and sketches future work.

II. Approach overview

The framework proposed in this paper is composed of three elements: (1) a modeling language for representing business processes with security concepts, (2) a modeling language for capturing procedural security policies, and (3) a verification engine which verifies the security policies against the business processes.

In our previous work, we proposed SecBPMN [22], a modeling language for modeling business processes with security concepts and security policies, that extends BPMN 1.1 with a set of security annotations defined by the Reference

978-1-5090-3694-3/16 $31.00 © 2016 IEEE
DOI 10.1109/REW.2016.11

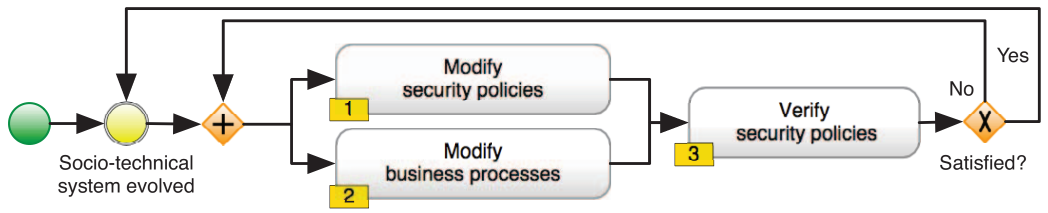

Fig. 1: Security verification process.

Model for Information Assurance and Security (RMIAS) [6]. The evaluation of SecBPMN underlined limitations of its expressiveness [22], [24] due to the limited set of concepts of BPMN 1.1 and lack of security concepts that could be represented.

In this paper we propose SecBPMN2, an extension of SecBPMN with BPMN 2.0, while adding to the first three new security annotations. In particular, we focused on BPMN 2.0 collaboration models: SecBPMN2 does not include conversation and choreography models, that are part of BPMN 2.0 standard. This decision is a consequence of the objective of creating a language as simple as possible. However, from our experience, conversations and choreographies models are less used than process and collaboration models.

The extension with all BPMN 2.0 collaboration elements greatly increases the expressiveness of SecBPMN adding the possibility of modeling numerous types of tasks and events, among many other concepts[1].

Following the naming convention of SecBPMN, we call the part of SecBPMN2 for business processes SecBPMN2- modeling language (SecBPMN2-ml), while we call SecBPMN2-Query (SecBPMN2-Q) the part of SecBPMN2 for security policies.

Inputs and output. The framework receives as input a document containing the description of running business processes with security concepts, and a specification of the security policies to which the system shall be compliant with. The framework returns a positive answer in case the policies are satisfied by the business processes.

Roles. The framework aims at helping security requirement engineers (a role that encapsulates the expertise of requirements analysts and security engineers or experts), in assisting them after the deployment of the socio-technical system to align business processes with security policies in light of evolution.

Process. We propose the process, shown in Fig. 1, to maintain security policies satisfied in business processes. The process starts with the design of business processes. Security policies (activity 1) and business processes are created (activity 2). After that, the software engine verifies security policies against business processes (activity 3). If the result is negative the business processes or the security policies are modified (activities 1-2) and the verification is executed again (activity 3). If the result is positive (they are satisfied), the business processes are deployed. After that, the process will remain inactive waiting for a new change in the system.

[1]For further details on BPMN 2.0 refer to [14].

III. Modeling Secure Business Processes and Security Policies

In this section, we formally specify the abstract syntax of SecBPMN2. Since SecBPMN2 is based on BPMN 2.0, we propose a formalization of the part of the standard that we need for the formal specification of the verification of business processes against security policies.

We employ the following notation: variables are strings in italic with a leading non-capital letter (e.g., x, y); sets are strings in the calligraphic font for mathematical expressions (e.g., $\mathcal{G}$, $\mathcal{I}$); relations and functions names are in sans-serif with a leading non-capital letter (e.g., $\mathsf{enforces}$); constants are in typewriter style with a leading non-capital letter (e.g., `true`, `false`). We use subscript to specify the tuple the elements belongs. For example, $\mathcal{A}_x$ indicates that the set $\mathcal{A}$ belongs to the tuple x. Relations are defined as set of ordered pairs, while functions are defined using functional notation.

Fig. 2 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 pool, namely Tower control operator and Pilots (that are further divided in lanes called Captain and Co-pilot), and contains at least one activity, e.g. Analyze request take-off. The order of execution of activities is represented with sequence flow relations that links two SecBPMN2 elements and specifies that the source element is executed before the target element. Gateways represent branches in the sequence 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. 2 specifies that the sequence flow can take two different paths, based on the evaluation of the condition Errors?. Security concepts are specified with security annotations, which are represented with orange solid circles, that are explained later in this section.

A. Formalization of BPMN 2.0

We define a BPMN 2.0 business process as tuple that contains the sequence of elements that can be executed, data objects, participants and association between them.

Definition 1 (BPMN 2.0 Business Process). *A BPMN 2.0 business process is a tuple*

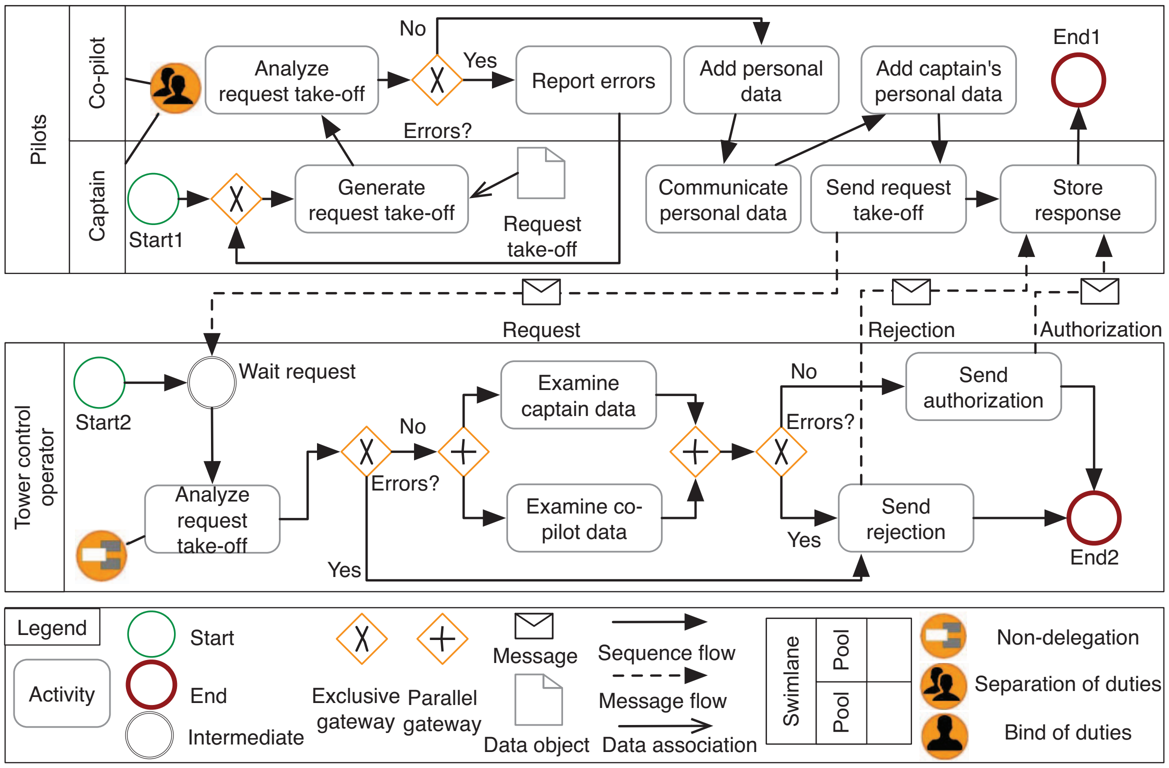

Fig. 2: Example of a SecBPMN2-ml collaboration model

$(\mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{D}, \mathcal{P}, \mathsf{sequenceFlow}, \mathsf{dataAssociation}, \mathsf{executor})$ *where:*

(a) $\mathcal{A}$ *is a finite set of activities,*
(b) $\mathcal{E} \subseteq \mathcal{E}^s \cup \mathcal{E}^e \cup \mathcal{E}^i$ *is a finite set of events,* $\mathcal{E}^s \cap \mathcal{E}^e \cap \mathcal{E}^i = \emptyset$*,*
(c) $\mathcal{G}$ *is a finite set of gateways,*
(d) $\mathcal{D}$ *is a finite set of data objects,*
(e) $\mathcal{P} \subseteq \mathcal{P}^{pool} \cup \mathcal{P}^{lane}$ *is a finite set of participants,* $\mathcal{P}^{pool} \cap \mathcal{P}^{lane} = \emptyset$*,*
(f) $\mathsf{sequenceFlow} \subseteq (\mathcal{A} \cup \mathcal{G} \cup \mathcal{E} \setminus \mathcal{E}^s) \times (\mathcal{A} \cup \mathcal{G} \cup \mathcal{E} \setminus \mathcal{E}^e)$ *is the sequence flow association,*
(g) $\mathsf{dataAssociation} \subseteq \mathcal{D} \times \mathcal{A} \times \{\mathtt{input}, \mathtt{output}\}$ *is the data association,*
(h) $\mathsf{executor} \subseteq \mathcal{P} \times (\mathcal{A} \cup \mathcal{E} \cup \mathcal{G})$ *is the executor association.*

Where: *(a)* is a set of activities $\mathcal{A}$; *(b)* is a set of events $\mathcal{E}$ that contains: a set of start events $\mathcal{E}^s$, that specifies where the business processes start; a set of end events $\mathcal{E}^e$, that indicates when the business processes finish; a set of intermediate events $\mathcal{E}^i$, that specifies when the business processes wait events to happen. The set $\mathcal{G}$, point *(c)*, contains the gateways, i.e., elements used to create variants in the sequence of activities executed. The set $\mathcal{D}$, point *(d)*, contains the set of data objects, i.e., the documents needed as input or outputted by activities, while the set $\mathcal{P}$, point *(e)*, contains the participants, i.e., the actors who execute the activities. Participants are divided in pools $\mathcal{P}^{pool}$, i.e., independent actors, and lanes $\mathcal{P}^{lane}$, i.e., sub-division of pools. The sequenceFlow association, point *(f)*, connects executable elements, i.e., $\mathcal{A}, \mathcal{E}$ and $\mathcal{G}$; from now on $\mathcal{AEG} = \mathcal{A} \cup \mathcal{E} \cup \mathcal{G}$. Association dataAssociation, point *(g)*, connects data objects with activities, and specifies if data objects are used as input or are outputted by the activities. Association executor connects executable elements to the participant who executes them. Such association is implicit in BPMN 2.0, when an executable element is inside a pool or a lane.

Following Definition 1, the business process executed by the participant Pilot in Fig. 2 is defined as a tuple with $\mathcal{A}$={Generate request take-off, Analyze request take-off, ... }, $\mathcal{E}^s$={Start1}, $\mathcal{E}^e$ = {End1}, $\mathcal{G}$= {Errors?}. $\mathcal{D}$ ={Request take-off} and $\mathcal{P}^{pool}$ = {Pilot} $\mathcal{P}^{lane}$={Captain, Co-pilot}. In Fig. 2 sequenceFlow connects, for example, Generate request take-off with Analyze request take-off; dataAssociation connects Request take-off with activity Generate request take-off and the value input. Association executor correlates, for example, Captain with Generate request take-off activity, meaning that the Captain executes the activity Generate request take-off.

Definition 2 specifies the conditions for a well-formed business process.

Definition 2 (Well-formed business process).
A BPMN 2.0 business process $(\mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{D}, \mathcal{P}, \mathsf{sequenceFlow}, \mathsf{dataAssociation}, \mathsf{executor})$ *is*

well-formed iff:

(a) $|\mathcal{E}^s| = 1$,
(b) $|\mathcal{P}^{pool}| = 1$, *and*
(c) $\forall x. x \in \mathcal{AEG} \rightarrow \; \exists!\, p \in \mathcal{P}^{pool}.(x,p) \in \mathsf{executor}$.

Where: *(a)* it has one start event; *(b)* it has one pool; *(c)* all executable elements are executed by only one pool. Points (b) and (c) are derived from BPMN 2.0 standard: they prevent a business process to be executed by multiple pools. Point (a) is needed for the automated verification, however, it does not limit the expressiveness because multiple start events can be substituted with a single start event, linked to a parallel gateway, in turn linked to all the activities connected to the original start events.

Business processes in Fig. 2, for example, are well formed because they have 1 start event each, 1 pool each, and all activities are associated with a single pool.

Collaboration models are the core part of BPMN 2.0, they allow modeling more business processes and the communications between them. In such models communications 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 Send request take-off creates a communication channel from the Pilots to the Tower control operator where the Request message is sent.

Definition 3 formally specifies a collaboration model: a set of business processes that exchange messages through message flows.

Definition 3 (BPMN 2.0 collaboration model).
A BPMN 2.0 collaboration model is a tuple $(\mathcal{BP}, \mathcal{M}, \mathcal{MF})$ *where:*

(a) $\mathcal{BP} = \{bp_1 ... bp_n\}$ *is a finite set of business processes in which*
$bp_i = (\mathcal{A}_i, \mathcal{E}_i, \mathcal{G}_i, \mathcal{D}_i, \mathcal{P}_i, \mathsf{sequenceFlow}_i, \mathsf{dataAssociation}_i, \mathsf{executor}_i)$
is the i-th bp in $\mathcal{BP}$ *and* $1 \leq i \leq n = |\mathcal{BP}|$,
(b) $\mathcal{M}$ *is a finite set of messages,*
(c) $\mathsf{MF} \subseteq (\mathcal{A}_i \cup \mathcal{P}_i^{pool}) \times (\mathcal{A}_j \cup \mathcal{P}_j^{pool} \cup \mathcal{E}_j) \times (\mathcal{M})$ *is the message flow relation, where* $i \neq j$ *and* $1 \leq i \leq |\mathcal{BP}|$ *and* $1 \leq j \leq |\mathcal{BP}|$, *and*
(d) $\mathsf{L} \subseteq \mathcal{AEG} \cup \mathcal{D} \cup \mathcal{P} \cup \mathcal{M} \times l$ *is a labeling relation.*

Where: *(a)* is a set of business processes; *(b)* is a set of messages; *(c)* is a message flow relation that represents the communications between business processes. A message flow links a pool or an activity of a business process to a pool, an activity or an event of another business process, to a message that is exchanged. In the collaboration model a labeling relation, *(d)*, associates a string to executable elements, data objects, messages and participants. In Fig. 2 two business processes are connected with three message flows that transmit the messages Request, Rejection and Authorization.

B. SecBPMN2 - modeling language

SecBPMN permits to specify eight security annotations defined in RMIAS [6], namely: accountability, auditability, authenticity, availability, confidentiality, integrity, non-repudiation and privacy. Security annotations are represented with a solid orange circle, with a black icon, that changes depending on the type of security annotation. We have added 3 security annotations, namely separation of duties, bind of duties and non-delegation. The added security annotations originated from our collaboration with security experts and practitioners, and from our experience of applying the language to numerous case studies. For some security annotations it is possible to specify attributes, called security properties, to specify further details.

Separation of duties is a security principle used to formulate multi-person control policies, requiring two or more different people to be responsible for the completion of a task or set of related tasks [27]. If the set of people changes during the process execution, separation of duties is dynamic, otherwise, separation of duties is static. Such security principle is represented with Separation of duties security annotations, while the static/dynamic property is represented with a boolean security property called "dynamic". For example, in Fig. 2 a separation of duties security annotation is linked to Captain and Co-pilots because a person cannot execute simultaneously the activities of captain and co-pilot. In this case we have dynamic separation of duties because, a person, in different flights, can be 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]. Bind of duties can be either static or dynamic. Such security concept is represented by the Bind of duties security annotation and it shares the same security property of separation of duties security annotation.

Non-delegation requires that a set of actions shall be executed only by the users assigned. It is represented by the Non-delegation security annotation. For example, in the business process in Fig. 2, since Non-delegation is linked to Analyze request take-off, the activity will be executed only by the Tower control operator, and no one else.

Definition 4 specifies a SecBPMN2-ml collaboration model as an extension of a BPMN 2.0 collaboration model with security annotations.

Definition 4 (SecBPMN2-ml collaboration model).
A SecBPMN2-ml collaboration model is a tuple $(CM, \mathcal{SA}, \mathsf{SecAss}, \mathsf{SAType})$ *where:*

(a) CM *is a collaboration model,*
(b) $\mathcal{SA}$ *is a finite set of security annotations,*
(c) $\mathsf{SecAss} \subseteq (\mathcal{SA}) \times (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i^{pool} \cup \mathcal{M} \cup \mathsf{MF})$ *, and*
(d) $\mathsf{SAType} \subseteq SA \times$ *{ Accountability, Auditability, Authenticity, Availability, Confidentiality, Integrity, NonRepudiation, Privacy, BindOfDuties, SeparationOfDuties, NonDelegation } is a relation that associates a type to each security annotation.*

Where: *(a)* is a BPMN 2.0 collaboration model; *(b)* is a set of security annotations; *(c)* is a security association relation that connects each security association with an element among activities, data objects, pools, message flows and messages; *(d)* is an association which is used to determine the type of the security association. For example, in Fig. 2, $\mathcal{SA}$ is composed of two security annotations and SecAss contains the link between the security associations and the elements of the collaboration, i.e., the links between the bind of duties security association and Captain and Co-pilot lanes and between the Non-delegation security annotation and Analyze request take-off activity.

We consider a SecBPMN2-ml collaboration model well-formed when all security annotations are correctly associated with BPMN 2.0 elements.

Definition 5 (Well-formed SecBPMN2-ml collaboration model).
A SecBPMN2-ml collaboration model $(CM, \mathcal{SA}, \mathsf{SecAss})$ *is well-formed iff:*

(a) $\forall \mathsf{bp}.\mathsf{bp} \in \mathcal{BP} \rightarrow \mathsf{bp}$ *is well formed,*

(b) $\forall (sa, t_1).((sa, t_1) \in \mathsf{SecAss} \wedge \mathsf{SAType}(sa) \neq \mathtt{BindOfDuties} \wedge \mathsf{SAType}(sa) \neq \mathtt{SeparationOfDuties}) \rightarrow \nexists (sa, t_2).(sa, t_2) \in \mathsf{SecAss}.t_1 = t_2$*, and*

(c) $\forall (\mathsf{sa}, \mathsf{t}).(\mathsf{sa}, \mathsf{t}) \in \mathsf{SecAss} \rightarrow$

(i) *if* $\mathsf{SAType}(sa) = \mathtt{Accountability}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i)$,

(ii) *if* $\mathsf{SAType}(sa) = \mathtt{Auditability}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i \cup \mathcal{MF})$,

(iii) *if* $\mathsf{SAType}(sa) = \mathtt{Authenticity}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i \cup \mathcal{M})$,

(iv) *if* $\mathsf{SAType}(sa) = \mathtt{Availability}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \mathcal{MF})$,

(v) *if* $\mathsf{SAType}(sa) = \mathtt{Confidentiality}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \mathcal{MF})$,

(vi) *if* $\mathsf{SAType}(sa) = \mathtt{Integrity}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \mathcal{MF})$,

(vii) *if* $\mathsf{SAType}(sa) = \mathtt{NonRepudiation}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{E}_i \cup \mathcal{MF})$,

(viii) *if* $\mathsf{SAType}(sa) = \mathtt{Privacy}$ *then* $t \in (\bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{D}_i \cup \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{E}_i)$,

(ix) *if* $\mathsf{SAType}(sa) = \mathtt{BindOfDuties}$ *then* $t \in \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i^{pool}$,

(x) *if* $\mathsf{SAType}(sa) = \mathtt{SeparationOfDuties}$ *then* $t \in \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{P}_i^{pool}$*, and*

(xi) *if* $\mathsf{SAType}(sa) = \mathtt{NonDelegation}$ *then* $t \in \bigcup_{i=1}^{|\mathcal{BP}|} \mathcal{A}_i$.

Where: *(a)* all the business processes in the SecBPMN2-ml collaboration model are well formed; *(b)* each security annotation, that is not bind of duties or separation of duties, is linked to one and only one element of the business processes; *(c)* security annotations are associated with a subset of BPMN 2.0 elements. Bind of duties and separation of duties can be linked only to pools, defined in *(ix)* and *(x)*, because they specify properties on performers of business processes. While non-delegation, point *(xi)*, can be associated only to activities: the only element of BPMN 2.0 that can be delegated to another participant. For space limitations we do not describe the constraints on the security annotations previously defined in SecBPMN, a complete description can be found in [22].

C. SecBPMN2-Query

SecBPMN2-Q is a modeling language expressly thought for the specification of procedural security policies. It builds on top of SecBPMN-Q [22], a query language (part of SecBPMN) that differently from other approaches [4], [7] is aimed at verifying security policies over business process models. SecBPMN2-Q extends SecBPMN-Q with BPMN 2.0 and the vast range of security annotations of SecBPMN2-ml.

SecBPMN2-Q extends SecBPMN2-ml with three relations: negative flows, walk and negative walk. All of them connect two SecBPMN2-ml elements among activities, events or gateways. **Negative flow relation** matches all business processes in which the target element is never executed right after the source element. **Walk relation** matches all business process in which the target element is executed after the source element. **Negative walk relation** is the negation of walk 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.

A SecBPMN2-Q security policy represents a class of business processes, therefore, its verification consists in checking if a business process fits in the class defined by the security policy. With the relations introduced in SecBPMN2-Q and the usage of “@" it is possible to define complex constraints and model a vast range of security policies.

Fig. 3 shows an example of SecBPMN2-Q security policy that matches all business processes in which: (a) the Tower control operator executes Analyze request take off, and it does not delegate the activity or part of it to other participants; (b) after that it sends an Authorization message to any participants who executes Store response.

Definition 6 specifies a SecBPMN2-Q security policy as SecBPMN2-ml collaboration model with negative sequence flow, walk and negative walk relations.

Definition 6 (SecBPMN2-Q security policy).
A SecBPMN2-Q security policy is a tuple $(SecCM, \mathsf{NSF}, \mathsf{Walk}, \mathsf{NWalk})$ *where:*

(a) $SecCM$ *is a SecBPMN2-ml collaboration model,*

(b) $\mathsf{NSF} \subseteq (\mathcal{AEG}_i) \times (\mathcal{AEG}_i)$ *with* $1 < i < |\mathcal{BP}|$,

(c) $\mathsf{Walk} \subseteq (\mathcal{AEG}_i) \times (\mathcal{AEG}_i)$ *with* $1 < i < |\mathcal{BP}|$*, and*

(d) $\mathsf{NWalk} \subseteq (\mathcal{AEG}_i) \times (\mathcal{AEG}_i)$ *with* $1 < i < |\mathcal{BP}|$.

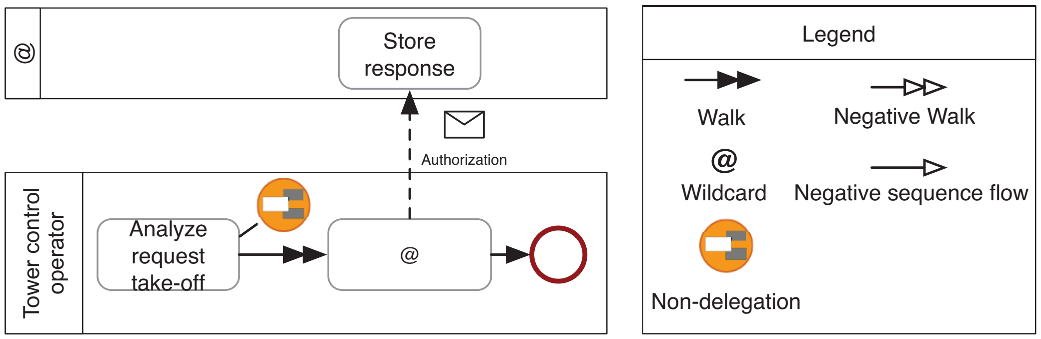

Fig. 3: Examples of a SecBPMN2-Q security policy.

Where: *(a)* is a secBPMN2 collaboration model; *(b)* is a Negative Sequence Flow (NSF) relation; *(c)* is a walk relation; *(d)* is a negative walk (NWalk) relation. Negative sequence flow, walk and negative walk relations connect two elements of the same business process, this is indicated with the same variable in the subscript of the elements of the ordered couples.

IV. Verification of Security Policies

The verification of security policies consists in checking if business processes satisfy the constraints defined in the policies. To perform this check, we defined a label correspondence function, to check whether the constraints in the security policies can be applied to a business process, a walk existence functions, to check if a walk connects two SecBPMN2 elements, and a security enforcement verification, which checks if a security annotation of a policy is enforced in a business process.

Definition 7 specifies the correspond function, which checks if two elements of SecBPMN2 have compatible labels and they are of the same type.

Definition 7 (Label correspondence check).
$\mathsf{correspond}:$
$(\mathcal{AEG}\cup\mathcal{D}\cup\mathcal{M}\cup\mathcal{P})\times(\mathcal{AEG}\cup\mathcal{D}\cup\mathcal{M}\cup\mathcal{P})\rightarrow\{\mathtt{true},\mathtt{false}\}$
$elem_{pol}, elem_{coll}\mapsto \mathtt{true}$ *iff:*

(a) $(elem_{pol}$ *and* $\mathsf{elem}_{\mathsf{coll}})$ *are elements of the same type, and*
(b) $\mathsf{L}(elem_{pol})=\mathsf{L}(elem_{coll})\vee\mathsf{L}(elem_{pol})$ = *"@".*

In Definition 7 we check if: *(a)* two elements are of the same type; *(b)* if the labels of the two elements are identical or if the label of the security policy contains a "@" and, therefore, it matches every elements.

The oneWalk function in Definition 8 checks if there exists at least one walk between two elements in a collaboration model.

Definition 8 (Check walk existence).
$\mathsf{oneWalk}: Walk\times SecBPMN2-ml\rightarrow\{\mathtt{true},\mathtt{false}\}$
$(s,t), coll\mapsto\mathtt{true}$ *iff:*

(a) $\exists x.x\in\mathcal{AEG}_{coll}.\mathsf{correspond}(x,s)$,
(b) $\exists y.y\in\mathcal{AEG}_{coll}.\mathsf{correspond}(y,t)\wedge x\neq y$, *and*
(c) *exists a walk from x to y in coll.*

Where oneWalk function returns true if: *(a)*, *(b)* there exists two distinct, correspondent elements in the SecBPMN2 collaboration model; *(c)* the elements are connected by at least a walk. The walk concept is not specified since is a well-known concept in graph theory [29].

Definition 9 defines a function that verifies if the security annotations of the policy specify stricter constraints than the security annotations of the business process.

Definition 9 (Security annotation enforcement verification).
$\mathsf{enforces}:$
$(SecBPMN2-Q\times SecBPMN2-ml\rightarrow\{true,false\}$
$pol, coll\mapsto\mathtt{true}$ *iff:*
$\forall(sa_{pol},elem_{pol}).(sa_{pol},elem_{pol})\in\mathsf{SecAss}_{pol}\rightarrow$
$\exists(sa_{coll},elem_{coll}).(sa_{coll},elem_{coll})\in\mathsf{SecAss}_{coll}.$
$\mathsf{correspond}(elem_{pol},elem_{coll})\wedge\mathsf{SAType}(sa_{pol})=$
$\mathsf{SAType}(sa_{coll})\wedge$

(a) $\mathsf{SAType}(sa_{pol})=\mathtt{BindOfDuties}\vee\mathsf{SAType}(sa_{pol})=$ $\mathtt{SeparationOfDuties}\rightarrow$
   (i) $\mathsf{dynamic}(sa_{pol})\leftrightarrow\mathsf{dynamic}(sa_{coll})$, *and*
   (ii) sa_{pol} *and* sa_{pol} *are connected to the same elements;*
(b) *[check enforcement of SecBPMN security annotations]*

Definition 9 specifies that all security annotations in a security policy are enforced if, for all of them, there exists at least one security annotation in the SecBPMN2-ml collaboration model that is connected to a correspondent element, see Definition 7, and has stricter security properties. Definition 9 defines the checks for the security annotations we introduced in this paper, for the security annotations previously defined for SecBPMN, please refer to [22]. For bind of duties and separation of duties, point *(a)*, the value of dynamic security property shall be the same, we use `dynamic` functions that returns `true` if dynamic security property is set to true, (point *(i)*); and the two security policies shall be connected to the same elements, point *(ii)*.

Definition 10 specifies a function that verifies a SecBPMN2-Q security policy against a SecBPMN2-ml collaboration model.

Definition 10 (Security policy verification).
$\mathsf{verifySecurityPolicy}:$

$SecBPMN2-Q \times SecBPMN2-ml \rightarrow \{\texttt{true}, \texttt{false}\}$
$pol, coll \mapsto \texttt{true}$ *iff:*

(a) $\forall x. x \in \mathcal{AEG}_{pol} \cup (D)_{pol} \cup (M)_{pol} \cup (P)_{pol} \exists y \in \mathcal{AEG}_{coll} \cup (D)_{coll} \cup (M)_{coll} \cup (P)_{coll}.Correspond(x, y)$,
(b) $\forall (x, y).(x, y) \in \mathsf{sequenceFlow_{pol}} \rightarrow (x, y) \in \mathsf{sequenceFlow_{coll}}$,
(c) $\mathsf{enforces}(pol, coll)$,
(d) $\forall walk.walk \in \mathsf{Walk}_{pol} \rightarrow \mathsf{oneWalk}(walk, pol, coll)$,
(e) $\forall nWalk.nWalk \in \mathsf{NWalk}_{pol} \rightarrow oneWalk(nWalk, coll) = \texttt{false}$, *and*
(f) $\forall (x, y).(x, y) \in \mathsf{NSF}_{pol} \rightarrow \not\exists (x_1, y_1).(x_1, y_1) \in \mathsf{sequenceFlow}_{coll}.correspond(x, x_1) \wedge correspond(y, y_1)$.

A SecBPMN2-Q security policy is verified in a SecBPMN2-ml collaboration model if: *(a)* for all the elements of the security policy there exists at least one elements with a correspondent label; *(b)* for all sequence flow relations in the security policy there exists a sequence flow in the collaboration model between two correspondent elements; *(c)* all security annotations of the security policy are enforced; *(d)* for each walk relation of the security policy there exists a walk in the collaboration model; *(e)* for each negative walk relation of the policy no walk connects the two elements; *(f)* for each negative sequence flow there is no correspondent sequence flow in the collaboration model.

V. Automated verification with STS-Tool

We developed STS-Tool [25] a software that supports the framework proposed in this paper. STS-Tool contains graphical editors that permit to model SecBPMN2-ml business processes and SecBPMN2-Q security policies. The software is written in Java, it is based on the Eclipse framework[2] and on the Graphical Editing Framework (GEF) [3].

STS-Tool integrates SecBPMN2, and the framework describe in this paper with a wider framework, which helps security analyst from the definition of early security requirements until their implementation [23], [20]. Such integration increases the range of possible applications of SecBPMN2.

When dealing with real-world scenarios, business processes can be considerably large, with hundreds of elements and tens of participants. Hence automating the verification of security policies against business processes becomes essential. We developed, and integrated in STS-Tool, a software engine on top of a planner for verifying business processes with security choices against security policies. Planners are software engines that generate plans, i.e., sets of actions, that respect a set of constraints and achieve a set of goals. We chose planners over Petri nets and model-checking based approached, because of their greater expressiveness.

We generate set of constraints from a business process so that the possible plans generated are the possible executions of the business process. For example, the business process in Fig. 2 executed by Co-pilot is transformed to a set of constraints for which the generated plans 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, on the other hand, are transformed to goals, i.e., in predicates that must be true once the plan is executed. For example, the policy in Fig. 3 is transformed to a goal that specifies that: (i) the message Authorization is sent from any activity executed by Tower control operator; (ii) Analyze request take-off is executed before the message is sent; (iii) Analyze request take-off is linked to a non-delegation security annotation. Thus, if is possible to find a plan that reaches the goals, then the business process satisfies the security policy.

We use $\mathcal{K}$ [8] as a planning language to define such constraints and goals, which is supported by $DLV^{\mathcal{K}}$ [8], a software that generates plans from a $\mathcal{K}$ specification.

Any Task of SecBPMN2 is transformed into a $\mathcal{K}$ predicate task(<Element name>) with a parameter which is the name of the element.

An Exclusive gateway is transformed into gatewayExclusive(<Element name>) predicate while all other gateways are transformed into inclusiveGateway (<Element name>) predicate, since parallel, event based and complex gateways behave as inclusive gateways, i.e., is possible to execute more than one outgoing sequence flow.

All Events are transformed into event(<Element name>) predicate. We do not distinguish between different types of events since all type of events have the same impact in terms of sequence of elements executed in a business process.

Data objects, Data reference, Data input and Data output are transformed into Data object(<Element name>) predicate, which represents the physical storage of information. Messages are transformed in the predicate message(<Element name>).

Participants, both lanes and pools are transformed into participant(<Element name>) predicate.

The sequence flow relation is transformed into a predicate sequenceFlow(<Source>, <Destination>) where the first parameter is the element source of the sequence flow, and the second element the target of the sequence flow.

The Message flow relation is transformed into the predicate messageFlow(<Source>,<Destination>,<Message>). Similarly to the sequence flow predicate, this predicate has two parameters that specifies the source and the target of the relation, moreover it requires a third parameter which specifies which is the message sent with the message flow.

SecBPMN2 uses one Data association relation to specify if a data object is used by another element and the direction of the association (incoming or outgoing from the element) to specify if the data object is read or written. We generate two predicates depending if the data object is read, readDO(<Data object name>,<Element name>), or written, WriteDO(<Data object name>,<Element name>). The two predicates use the same parameters: the name of the data object and the name of the element which use the data object.

When an element is placed inside the scope of a participant an Ownership relation is specified between the two elements. We generate a similar predicate, i.e., owns(<Name pool>,<Element

[2]https://eclipse.org
[3]https://eclipse.org/gef/

name>), to specify the same concept in $\mathcal{K}$. The ownership relation is not limited to elements such as tasks and data objects, but binds together also lanes specified inside pools.

We generate a predicate for each type of security annotations, with the security properties specified as parameters.

We do not generate predicates for Artifacts, such as group and textual annotations, since, they are irrelevant for the execution of the business process.

Listing 1 shows an example of a $\mathcal{K}$ representation of a SecBPMN2-ml collaboration model. In $\mathcal{K}$ strings that start with an upper case letter are considered variables, while strings that start with a lower case letter are constants. The former represents any activity, while the latter represents specific activities. Lines 1-3 define the message and the sequence flow fluents. The sequence flow fluent requires two activities as parameters and represents the sequence flow between two elements. The message flow fluent requires as parameters two activities and a message, and represents the message flow between two activities. Lines 4-7 represent part of the collaboration model in Fig. 2. In particular, Lines 4-6 represent the sequence flow between the Add captain' personal data and Send request take-off, and the message flow containing the message Request between the latter activity and Analyze request take-off. Lines 7 specifies the security annotation linked to the activity Analyze request take-off.

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

Listing 1: A $\mathcal{K}$ specification of part of the SecBPMN2-ml business process in Fig. 2

Listing 2 shows the $\mathcal{K}$ goal generated from the security policy in Fig. 3. Lines 2-3 contain the predicate executed that indicates the activity specified in the first parameter must be executed by the participant indicated by the second parameter. For example Line 2 instructs the planner to find a plan in which the activity Analyze request take-off is executed by Tower control operator. Line 4 specifies that the plan will contain a message sent by any activity to Store response and transmitting the message Authorization. Lines 5 specifies that the activity Analyse request take-off shall be annotated with a non-delegation security annotation. Line 6 instructs the planner to find a plan where path1 is executed. path1 is defined as a set of constraints (Lines 8-9), where path1_1 becomes true after Analyze request take-off is executed and path1 becomes true if path1_1 is true and activity X is executed. Activity X is the same activity of Line 4, i.e., the one that sends the Authorization message to activity Store response. Line 7 specifies the maximum number of actions the generated plan shall contain; we set this as the number of all SecBPMN2-ml elements in the business process. This parameter does not influence the generated plans, if it is high enough, i.e., as high as the longest possible execution, which 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   path1
7   ? (10).
8 caused path1_1 after exec(analyze request take off).
9 caused path1 after exec(X), path1_1.
```

Listing 2: Example of a $\mathcal{K}$ goal generated from Fig. 3

VI. Evaluation

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

Findings from the case study. We evaluated the framework using the System Wide Information Management (SWIM) [9] Air Traffic Management (ATM) case study, which is a variant of the case study "*The emerging European Air Traffic Management systems*" of the project Aniketos [2].

We analyzed a set of requirement documents that specify SWIM ATM business processes, after which 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, in particular, 7 of them with at least a Separation of duties security annotation, 1 with at least a Bind of duties security annotation, and 6 security policies with at least one Non-delegation security annotation.

We followed the security verification process of Fig. 1 iterating multiple times the inner cycle. After we modeled the business processes and the security policies, we verified the latter against the former. A set of security policies were not satisfied, therefore, we analyzed again the documentation, in the light of the security issues highlighted by the verification, and we updated the business processes. The modifications of the business processes consisted in changing the types of tasks and events and in adding security policies. For what concerns the security annotations introduced in this paper, we added 2 Separation of duties security annotations and 4 Non-delegation security annotations. We iterated the cycle until all security policies were verified.

The framework allows fast iterations in the inner cycle in the process in Fig. 1. This permitted an incremental specification of the business process models and the security policies, facilitating the overall security analysis of the case study. The process, proposed in this paper, is easy to follow and can be used on all the stages of the software development life cycle. On the other hand the process is not related to any software engineering method, such as waterfall, incremental or spiral and, therefore, it may be not intuitive to understand when to use the framework.

SecBPMN2 was enough expressive to model all the business processes of the case study. The complexity of the modeling language, inherited from BPMN 2.0, may require a training session for inexperienced modelers. However, from our experience, once familiarized with BPMN 2.0, the effort required for learning how to use the security annotations and the relations introduced in SecBPMN2 is minimal.

Scalability evaluation. We performed a scalability study to evaluate how well the verification software engine performs overs large models and to find which factors of the problem, i.e. elements of SecBPMN2-ml, influence its performance. We used time as a metric to test the scalability of the software. We identified 8 factors that increase the complexity of the problem: the number of activities, participants, data objects, events, processes, paths, message flows, and that of security annotations.

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, which is polynomial, and for the number of processes, which is exponential. Fig. 4 shows the results of the tests about the number of activities, while Fig. 5 shows the results for the number of the processes. The results indicate that the framework scales well: the exponential grown related to the number of processes does not influence significantly the overall scalability because, from our experience, even in complex 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].

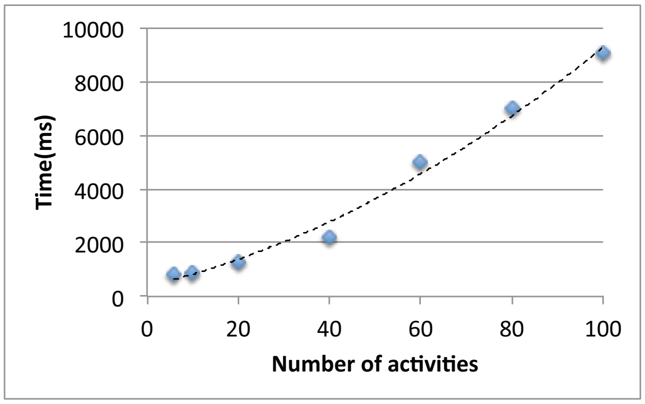

Fig. 4: Test results on number of activities.

VII. Related work

As far as our knowledge goes, there are no approaches that cover the overall process proposed in this paper. In the following, we describe the most known approaches that deal with the verification of (security) requirements in business processes.

Modeling business processes with security concepts. Many graphical modeling languages extending BPMN [14] with security aspects have been proposed. SecureBPMN [5], proposed by Brucker et al., captures security and compliance requirements. Other extensions of BPMN also rely on security annotated business process modelling [30], [19], [13] similarly to SecBPMN2. However, differently from existing approaches, ours allows the definition of custom security policies. Instead, existing approaches, e.g. [30], [21], employ software engines that check a fixed set of hard-coded security policies.

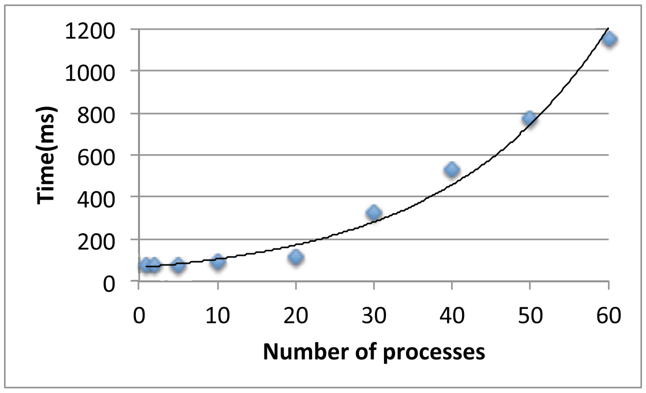

Fig. 5: Test results on number of processes

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) [4] and BPQL (Business Process Query Language) [7] allow to graphically define queries and check which business processes satisfy the queries. These two query languages are not based on BPMN 2.0, 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.

Other approaches are built on formal mathematical concepts (e.g. first order or temporal logic), and can be used to define business processes and/or queries. These languages are expressive enough to include in the model security concepts. For instance, the approach of Rushby [18] proposes a language and a framework that checks if the code of the software diverges from specified behaviors (i.e., policies). The main drawback of these approaches is low usability, since they are quite complex and require lot of effort for the formalization of both business processes and security policies. In light of real-world scenarios, whose dimensions get larger and larger, it is nearly impossible to model business processes with such languages.

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, as they require a substantial effort for formalizing business processes and security policies. Ghose and Koliadis [11] enrich BPMN with annotations, and calculate how much a business process deviates from another one. Differently from our approach, theirs focuses only on the structural differences with no consideration of security requirements. Other works [3], [17] 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 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) [10] 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) [16] and Planning Domain Definition Language (PDDL) [12] 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.

VIII. Conclusions and future work

We have proposed a framework for maintaining the security policies satisfied in business processes during the evolution of socio-technical systems. Every time an evolution step leads to a change in the business processes or in the security policies, the framework verifies the former against the latter and possible security issues are solved.

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 framework with a case study from the air traffic management domain.

Although mature and quite comprehensive, the framework suffers from these limitations: (i) the SecBPMN2 is built on top of BPMN 2.0, and inherits the complexity of the language; (ii) limited scalability of the software engine, due to the computational time that grows exponentially with the number of processes.

Future work consider: (i) extending the modeling languages to allow defining customizable security annotations, and (ii) empirical evaluation of the framework with practitioners, where they serve as modelers and decision makers.

Acknowledgement This research was partially supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 653642 - VisiON.

References

[1] Managing cyber risks in an interconnected world: Key findings from the global state of information security survey 2015. Technical report, PWC, September 2014.
[2] Aniketos Website. Last visited October '15. http://aniketos.eu.
[3] 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.
[4] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Information Systems*, 33(6):477–507, 2008.
[5] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In *SACMAT '12*.
[6] Y. Cherdantseva and J. Hilton. A Reference Model of Information Assurance and Security. In *Proc. of ARES*, pages 546–555, 2013.
[7] D. Deutch and T. Milo. Querying Structural and Behavioral Properties of Business Processes. In *DPL '07*, pages 169–185.
[8] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete knowledge. In *CL '00*, pages 807–821.
[9] Eurocontrol. System wide information management (swim), April 2013.
[10] 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.
[11] A. Ghose and G. Koliadis. Auditing Business Process Compliance. In *ISOC '07*.
[12] 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.
[13] M. Menzel, I. Thomas, and C. Meinel. Security Requirements Specification in Service-Oriented Business Process Management. In *ARES '09*, pages 41–48.
[14] OMG. BPMN 2.0, Jan 2011.
[15] E. Paja, F. Dalpiaz, and P. Giorgini. Managing Security Requirements Conflicts in Socio-Technical Systems. In *ER '13*, pages 270–283.
[16] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR '89*, pages 324–332.
[17] J. L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In *ICATPN '96*, pages 400–419.
[18] J. Rushby. Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises. *Reliability Engineering and System Safety*, 75:167–177, 2002.
[19] 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*.
[20] M. Salnitri, A. Brucker, and P. Giorgini. From Secure Business Process Models to Secure Artifact-Centric Specifications.
[21] M. Salnitri, F. Dalpiaz, and P. Giorgini. Aligning Service-Oriented Architectures with Security Requirements. In *OTM '12*, pages 232–249.
[22] M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and Verifying Security Policies in Business Processes. *BPMDS '14*, pages 200–214.
[23] M. Salnitri, E. Paja, and P. Giorgini. Preserving compliance with security requirements in socio-technical systems. In *Proc. of CSP*, pages 49–61, 2014.
[24] Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini. Designing secure business processes with secbpmn. *Software & Systems Modeling*, pages 1–21, 2015.
[25] Mattia Salnitri, Elda Paja, Mauro Poggianella, and Paolo Giorgini. Ststool 3.0: Maintaining security in socio-technical systems. pages 205–212, 2015.
[26] SecBPMN Website. Last visited October 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, 2003.
[29] R. Wilson. *Introduction to Graph Theory*. 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.