Abstract—Security is a central issue in socio-technical systems, i.e., large systems composed of autonomous subsystems that interact to achieve their objectives. The engineering of secure socio-technical systems is not just a technical challenge, but it needs to consider the social and organizational components of socio-technical systems too.

But, security specification is to no avail without the correct enforcement of such requirements. Security enforcement in socio-technical systems is no trivial task: the conceptual gap between social/organizational concepts and operational ones (mechanisms used in implementation), generates ambiguities that, combined with the size and complexity of such systems, may lead to an erroneous enforcement of security requirements.

In previous work we have proposed STS, a security requirements method for socio-technical systems. In this paper, we build on top of STS and its modeling language, STS-ml, to capture security requirements for socio-technical systems, and create a comprehensive framework to enforce them into the implementation. The framework proposes transformation rules that allow passing from social/organizational requirements to procedural models, to security policies, that once verified are mapped to enforcement rules used in the implementation, bridging the gap between high-level security requirements and security mechanisms of the implementation.

We apply our framework to a case study about e-commerce, and report on promising results from empirical evaluation.

I. INTRODUCTION

Most of today’s software systems are part of larger systems, that include not only technical (software and hardware) components, but also humans and organizations. These larger systems are socio-technical systems [32], [7], and they consist of autonomous subsystems (participants) that interact—by exchanging messages—to achieve their objectives. Examples include smart cities, e-commerce systems, online banking, etc. In an e-commerce socio-technical system, organizations such as e-commerce shops and banks interact with one another and with customers (being humans or organizations) to buy and sell goods through electronic payments.

Security is key to the engineering of socio-technical systems, because a security issue in a single component may affect other components in the system leading to severe consequences, such as loss of privacy and law infringement. For example, in the case of electronic payments, an unauthorized disclosure of sensitive data (due to security issues of an e-commerce shop), such as customers’ bank account details, may affect the shop’s customers and related banks.

The engineering of a secure socio-technical system—and of the participating secure software systems—requires a thorough analysis of social and organizational aspects, in addition to technical ones [19]. The analysis of the latter identifies security problems within the technical level (e.g., having low security protection on the server where the database is stored), or at the interplay of social and technical subsystems (e.g., a malicious user accessing the database where buyers’ records are kept or performing an SQL injection attack). A social/organizational analysis, on the other hand, allows capturing yet another range of security issues, such as, for instance, the need for a seller to have a buyer’s detailed purchase history (for customized offers) does not necessarily comply with the will of the buyer to keep this information private (to himself).

However, the specification of security requirements alone is not enough. Security requirements must be enforced in the implementation, to ensure that the said system satisfies them. But, there is a big gap from high-level security requirements to the security mechanisms and solutions that are part of the implementation. Most importantly, such implementation should account for the underlying business processes that support participants’ interactions.

Existing approaches either deal with social/organizational or technical security aspects separately [5], [4], or allow tracing security requirements till the implementation without providing any rules for their enforcement [12], [13]. Without well-defined transformation rules, one risks missing important details necessary for the correct enforcement of security requirements. For example, an erroneous transformation of the set of social actors allowed to access users’ information to technical participants in the implementation (e.g., roles in role-based access control) will lead to ill-defined authorizations and, consequently, to security issues in the implementation.

In this paper, we propose a comprehensive framework for the design of secure socio-technical systems, based on STS [6], which starts with the specification of socio-technical security requirements and is finalized with the enforcement of technical security mechanisms (supporting security requirements) in a skeleton implementation of the system-to-be. Our framework suggests to iteratively (i) create social/organizational models for the system-to-be, (ii) model procedural security aspects, (iii) verify security policies against procedural models, and finally (iv) enforce security.
The contributions of the paper are as follows: (i) a set of transformation rules to enforce social/organizational security requirements; (ii) analysis techniques to verify if security requirements are enforced in the design; (iii) an implementation of our framework, as part of a CASE tool called STS-Tool, that supports automated transformations, generations and analysis; (iv) results from interviews with security experts and practitioners on the need of such a framework, together with the results of an empirical experiment with modelers to show the effectiveness of the framework.

Section II provides an overview of the framework and the process that guides its application. Sections III-VII show the use of the framework illustrating each step of the process. Section VIII describes the automated analysis techniques and transformations implemented in the software tool, while Section IX exposes the results of the interviews and the empirical evaluation. Section X discusses the relevant related work, and finally Section XI concludes.

II. FRAMEWORK OVERVIEW

The wide scope of the framework, from the specification of security requirements to their implementation, calls for a flexible process to support its application. Given the complexity and size of socio-technical systems, the artifacts of the process are employed while models are created and analyzed iteratively and incrementally. The framework can be adopted in the context of broader methods for system engineering and for software engineering. Broadly, the adoption of the framework is appropriate for open and large-scale systems, for the social/organizational analysis is justifiable for socio-technical systems, and not for software systems (e.g., compilers).

Roles. The process is executed by a security requirement engineer, a role that encapsulates the expertise of a requirements analyst and a security engineer or expert. Such role requires a remarkable set of skills in modeling and security, therefore we suggest to execute the process by a team of experts.

Inputs/outputs. The process receives in input the requirements document (the result of elicitation activities), with a focus on security requirements. The final output of the process is a skeleton implementation of the technical systems (of the socio-technical system) that enforce security requirements.

Phases. Fig. 1 shows the overall process. In the following, we describe each phase, highlighting its importance.

Phase 1 supports the modeling of the socio-technical system under consideration. Such modeling focuses on representing the context, the main stakeholders (in terms of actors), their objectives, as well as their interactions. Most importantly, it captures security issues from stakeholders’ needs. This phase results in: (i) the creation of a diagram representing the social/organizational model of the system-to-be (STS-ml Diagram, Fig. 1); (ii) a list of social/organizational security requirements (STS-ml security requirements, Fig. 1).

Phase 2 deals with the transformation of social/organizational models to procedural ones, receiving in input a social/organizational diagram and generating a procedural diagram (SecBPMN2-ml Diagram, Fig. 1) of the system-to-be. This step is required, given that the level of detail contained in social/organizational models is not sufficient for the generation of the skeleton implementation. These models are at a high level of abstraction, and as such there is a big gap between security requirements specification and implementation. We use of procedural models to bridge this gap, for they bring in operational aspects, such as sequences of actions, security choices over the execution of such actions, etc.

Social/organizational models do not contain the information required to generate complete procedural models. Therefore, Phase 3 is executed to enrich the procedural models generated by phase 2 with details such as the temporal aspects and security choices on the executed processes.

Phases 1-3 are repeated until security requirement engineers decide they have captured all important (security) details and the procedural model is complete and accurate.

Phase 4 generates procedural security policies from social/organizational security requirements. It requires in input the list of social/organizational security requirements (STS-ml security requirements, Fig. 1) and generates procedural security policies (SecBPMN2-Q security policy, Fig. 1). This phase permits to translate security requirements, defined in terms of social/organizational concepts, in more operational constraints, as the ones specified in procedural security policies.

Phase 5 deals with the verification of procedural security policies, generated in phase 4, against the procedural model, enriched in phase 3. This phase validates the procedural models by verifying if they enforce the security requirements specified in the social/organizational model. If all security policies are enforced, then the security requirement engineers have the proof that the procedural model meets the security requirements. If the procedural model does not satisfy all procedural security policies, the process starts from the beginning.

Phase 6 consists in generating a skeleton of the implementation from the enriched and verified procedural model. The resulting application will enforce the social/organizational se-
Delegation requirements, because social/organizational security requirements define the security policies that are verified against the procedural model. Therefore, because the transformation enforces all the security aspects defined in the procedural model, the implementation can be considered secure.

The Example II.1 describes a socio-technical system we use through the rest of the paper to illustrate the phases in detail.

**Example II.1 (Payment Engine).** SAP Payment Engine (PE) [28] is a software system created to perform payments for e-commerce shops or, more in general, for services that allow users to pay with electronic methods. Usually, to support the electronic payments, the e-shops implement interfaces to communicate with all the banks they intend to use as source/destination of the payments. But each bank requires a different set of protocols and security measures. Therefore the e-shops are forced to put a noticeable amount of effort to implement different interfaces, and for medium/small companies it is not acceptable investing large quantity of time and money just to allow people to pay the goods/services they offer. The PE minimizes such effort: it contains a set of interfaces with the most known banks in the world that can be used out of the shelf. E-shop programmers only have to create one interface to transmit the required data to the PE system.

The payment system is a socio-technical system since it involves customers, banks, the PE, etc., which interact with each other to perform payments.

### III. Modeling Social/Organizational Security Aspects

We start with an analysis of social and organizational aspects, in addition to technical ones, for the system-to-be, which results in a requirements model and a security requirements specification for the given socio-technical system. For this, we use the STS-ml [5], [19] modeling language. STS-ml is an actor and goal-oriented security requirements modeling language for socio-technical systems. It was chosen because: (1) it is specifically thought for socio-technical systems, relating security to interaction, (2) it supports a rich set of security requirements, while providing a clearer set of security requirements than existing approaches [11], [16].

In STS-ml, requirements models are created through three views: (i) the social view—represents the main stakeholders (in terms of actors) together with their objectives (via goals) and the interactions they enter in the socio-technical system; (ii) the information view—represents stakeholders’ informational assets and their representation via documents; and (iii) the authorization view—represents the authorizations that actors grant to others over their information. Fig. 2 shows a partial STS-ml model of the motivating example.

**Social view.** Actors in STS-ml are modeled in terms of (i) agents—concrete entities that are already known at design-time (e.g., Payment Engine), and (ii) roles—abstract entities representing a class of participants (e.g., Bank dst). An actor’s rationale captures actors’ goals, and how they are achieved via AND/OR goal decompositions (e.g., the root goal of the Payment Engine is Value transferred). Moreover, to achieve their goals, actors might need to read or modify documents, as well as create (produce) new documents (e.g., Payment engine reads document Transfer order to achieve goal Value transferred). Most importantly, the social view captures actors’ social interactions via two social relationships: goal delegation and document transmission. STS-ml allows actors to express their concerns about security (security needs) over the interactions they enter to then derive security requirements with respect to confidentiality, integrity, availability, accountability, reliability, and authenticity.

**Information view.** Information is a first class citizen in STS-ml, since most security issues are concerned with the protection of information. Information owners are the ones concerned with the protection of information. Therefore, STS-ml allows specifying information ownership via the relationship own that relates an actor to the information it owns.
Information may be available in various forms, and thus, STS-ml distinguishes information from its representation in form of documents. Documents become relevant from a security point of view because of the information they represent. Thus, the purpose of the information view, apart from representing information entities and their respective owners, is to link together the documents actors use and exchange in the social view with their informational content. This link is drawn through Tangible by relationships. In Fig. 2, information e-transfer details is made tangible by document Transfer order.

Authorization view. An adequate representation of permissions and prohibitions is crucial to establishing whether information is used and exchanged in compliance with security requirements. The authorization view allows specifying the permissions and/or prohibitions on information that actors grant one to another. An authorization relationship details: (i) the permissions/prohibitions on the operations actors can perform over information (Read, Modify, Produce, Transmit) while manipulating documents for the achievement of their goals; (ii) information entities for which permissions/prohibitions are specified; (iii) the scope of authorisation, referring to the goal(s) for the fulfillment of which permission/prohibition is specified; and finally, (iv) transferability, specifying whether permissions can be further granted to others (not applicable to prohibitions). In Fig. 2, the Payment Engine authorizes the Bank dst to read e-transfer details in the scope of goal Dst transfer authorized, granting a transferable authorization. Security requirements are generated from authorizations whenever prohibitions are specified. For example, from the authorization to Bank dst, three security requirements are generated, one for each operation that is not authorized, namely: (i) non-modification, non-production and non-disclosure (i.e. not transmission) of information e-transfer details, see Fig. 2.

IV. MODELING PROCEDURAL SECURITY ASPECTS

Procedural models specify sequences of actions that are executed by socio-technical systems. They share enough details of such systems with social/organizational model to be generated from them, but at the same time, they specify operational details that could to be directly executed, e.g., WS-BPEL [17].

We use the SecBPMN2 (Secure BPMN 2.0) framework, an extension of SecBPMN [27] with BPMN 2.0 [18], to model procedural security aspects of socio-technical systems. The language is composed of: SecBPMN2-ml (SecBPMN2-modeling language), a language for modeling procedural aspects and security choices, and SecBPMN2-Q (SecBPMN2-Query), a graphical query language for specifying procedural security policies in terms of SecBPMN2-ml elements.

Other modeling languages have been proposed for specifying procedural aspects and security choices, such as those proposed by Menzel et al. [15], Rodriguez et al. [21] and Saleem et al. [24]. However, such modeling languages are either not designed to express security policies [15], or they permit to specify only a subset of the security aspects that can be expressed with SecBPMN2 [21], [24].

Fig. 3(a) 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 Payment Engine and Bank src, and contains at least one activity, e.g. Analyze transfer order. Each participant may own one or more data objects that represent physical entities in which information is stored. For example, the Payment Engine owns the data object Customer’s information. Data objects are read/written by one or more activities. The dashed arrow from Customer’s information data object to the activity Generate transfer document specifies that the activity reads the data object when it is executed. A dashed arrow in the opposite direction, i.e. from an activity to a data object, specifies that the activity writes the data object during its execution. 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 transfer order creates a communication channel from the Payment engine to the Bank src where the Transfer order is
sent. The order of execution of activities is represented with the message flow relation 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 transfer order in Fig. 3(a) specifies that the control flow can take two different paths, based on the evaluation of the condition Transfer order correct?: The exclusive gateway on the right merges the two branches of the control flow.

SecBPMN2-ml permits to specify security aspects in a business process with 11 security annotations that can be added and linked to SecBPMN2 elements. Security annotations are represented with a solid orange circle, with a black icon, that changes depending on the type of security annotation. For example, the security annotation associated to activity Accredit value in Fig. 3(a), called non-repudiation, specifies that the Bank src will not be able to legally repudiate the execution of the activity Accredit value. The list of all security annotations and their semantics can be found in [31].

V. FROM SOCIAL/ORGANIZATIONAL MODELS TO PROCEDURAL MODELS

The specification of procedural models from social/organizational models in not straightforward in real-world socio-technical systems, which tend to become large and complex. To facilitate this transition, we propose a set of transformation rules to generate SecBPMN2 business processes coherent with the STS-ml models.

The transformation is based on the mapping shown in Fig. 4(a,b). Social/organizational concepts (Fig. 4(a)), represented by STS-ml’s concepts, are linked to procedural concepts (Fig. 4(b)), represented by SecBPMN2’s. In particular, STS-ml concepts of role and agent are mapped to the SecBPMN2 concepts of swimlane and pool. Roles and agents represent autonomous active entities; similarly swimlanes and pools represent autonomous participants that execute processes. The STS-ml concept of goal is linked to the concept of process, because the execution of a process can achieve one or more goals. The concept of document in STS-ml is mapped to the concepts of data object and messages because documents, data objects and messages represent physical objects in which information is stored.

Such mapping permits to define complex transformation rules that allow transforming parts of STS-ml models to SecBPMN2-ml business process models. Unfortunately, there is not a commonly shared definition of how socio/organizational concepts can be transformed in procedural concepts. Such transformation rules change with respect to contexts, stakeholders, and security requirement engineers. For these reasons, we defined a language that permits to specify custom transformation rules. It merges STS-ml and SecBPMN2-ml with the distinction that the labels of the elements are used to bound STS-ml elements with SecBPMN2 elements.

Fig. 5(a) shows an example transformation rule. The upper part of the figure shows a part of an STS-ml model, namely a transmission, that will be transformed. Each element’s label starts with the keyword “$” followed by a unique string to identify the element. The lower part of the figure contains the SecBPMN2-ml model that is generated from the STS-ml model. The SecBPMN2-ml elements with the same label as the STS-ml elements will inherit the name once the transformation is executed. Fig. 5(b) shows an example of the application of this transformation rule. The upper part shows the transmission of document Transfer order from the Payment Engine to the Bank src, while the lower part shows the generated procedural model with Payment engine instead of “$B”, Bank src instead of “$A” and Transfer order instead of “$D”.

Multiple transformation rules can be defined for the same set of STS-ml elements. In the framework we define a set of default transformation rules that covers almost all possible transformations and allows to make transformation rules transparent to users when generating procedural models. Transformation rules can be applied only to the social view of STS-ml diagrams, because it describes the active part of the system, i.e., it contains STS-ml elements that can be transformed to one or more SecBPMN2-ml models. Instead, information in the authorization and information views will be used to specify security requirements.
VI. FROM SOCIAL/ORGANIZATIONAL SECURITY REQUIREMENTS TO PROCEDURAL SECURITY POLICIES

The forth and fifth phases of the process proposed in this paper consist in transforming social/organizational security requirements in security policies and verify them against procedural models. The forth phase requires defining a set of transformation rules similar to the ones defined for phase 2, with the difference that we pass from STS-ml security requirements to SecBPMN2-Q (SecBPMN2 - Query) models.

Fig. 3(b) shows an example of a SecBPMN2-Q procedural security policy. The arrow with two heads between *Analyze transfer order* and *Accredit value* is a Path which matches all the SecBPMN2-ml models where the first and the second activities are connected through an arbitrary long list of elements. The “@” wildcard, followed by any string, is used to match any SecBPMN elements of the same type. For example, the procedural model of Fig. 3(a) matches the procedural security policy of Fig. 3(b), because the activity *Send transfer order* (matches by the “@” activity in the security policy) sends the message *Transfer order* to the activity *Analyze transfer order*, which is connected with *Accredit value* through a gateway and the activity *Authorize transfer*.

We provide default transformation rules for all the social/organizational security requirements, leaving this transformation, too, transparent to the users. The STS-ml PE model in Fig. 2 specifies 12 security requirements, 4 specified in the social view, and 8 specified in the authorization view. Each security requirement can be transformed in one procedural security policies, using the default transformation rules we provided or a customized one.

The 5th phase of the process consists in verifying procedural security policies, created by phase 4, against procedural models. Since procedural security policies are generated from social/organizational security requirements, a positive verification means that the procedural model is as secure as specified in the social/organizational model of the socio-technical system.

VII. GENERATING A SKELETON OF THE IMPLEMENTATION

The generation of the implementation of a socio-technical system is the primary objective of the framework we are proposing. Generating a complete implementation of a system from a SecBPMN2 model is not possible, because the information contained in the models is not sufficient. Each activity in the procedural diagrams is identified with a short string whose interpretation changes with respect to the context and the reader of the diagram. As far as our knowledge goes, it is not possible to generate executable code from such short strings. For example, the activity *Store failure report* in Fig.3 can be interpreted as the action of saving the report in a data base, or creating a backup in a different storage unit, etc.

For such reasons, we chose to generate part of the implementation that is still central for most socio technical systems, but minimally relies on the functional part: the business artifacts, i.e. entities, data and documents [25].

---

We chose River Definition Language (RDL) [29] as the implementation language. RDL is an executable specification language that allows specifying declaratively: (i) the artifacts (e.g., entities); (ii) the relationships between them (e.g., associations); (iii) the business logic (e.g., actions) on the artifacts. RDL implementations generated with the transformation rules we provide, can be directly deployed. For further details please refer to [25].

Listing 1 illustrates an example of an RDL application, which allows to specify entities (e.g., customerAccount), their attributes (e.g., dateOfBirth), and custom types of the attributes (e.g. LocalDate). Besides this pure data modeling, RDL also supports specifying the actions (e.g., updateInformation) in a declarative style. Finally, RDL supports the specification of role-based access control restrictions: the actions of the entity customerAccount are only accessible by members of the role Consultant.

Fig. 4(b,c) shows the mapping between SecBPMN2 concepts and RDL concepts, which we use to define the transformation rules to generate RDL applications from SecBPMN2 models.

A Data object, which represents a set of information, is mapped to Type, which represents the structure of the information of an element in entity. A Message, that represents a set of information sent between pools, is linked to Type, which in this case represents the structure of a message sent in RDL.

A Pool, which defines a company or an actor such as a buyer or a manufacturer, is linked to an Application, that represents a set of business artifacts, which can be accessed only using the APIs, and their business logic. Both pools and a RDL applications are used to identify organizations or well-defined parts of them.

An Activity represents an operation performed by a participant; similarly, an Action represents the business logic linked to a data structure, i.e., they are the operations executed to set/maintain some properties of the business objects.

A Swimlane represents a participant, i.e., a person, a service or a set of them and it is mapped to Role, which represents any entity that can receive an authorization.

A Sub-process is a task that encapsulates a business process, which contains a set of SecBPMN2 elements. It is linked to a Call that is a reference to another RDL application, which, in turn, contains a set of business artifacts.

We defined in [25] a set of transformation rules, based on the mapping described above, to generate RDL applications that
enforce the security choices specified in procedural models. If the procedural models verify security policies generated from social/organizational security requirements, the generated applications meet the initial security requirements.

VIII. AUTOMATED ANALYSIS AND GENERATION

We created a software tool that supports the security requirement engineers in using the proposed framework. The tool allows to: (i) draw social/organizational and procedural diagrams, using STS/ml and SecBPMN2 respectively; (ii) to easily switch between diagrams; (iii) to verify the consistency of the models; (iv) generate procedural security policies; (v) verify such policies; and (vi) to generate a skeleton of the implementation of socio-technical systems.

A. Consistency verification

Checking if the diagrams are syntactically correct is of a great support to modelers, for it helps modelers to create well-formed diagrams.

This feature also includes the verification of social/organizational security requirements against the social/organizational model. If security requirements are not verified, the models derived from the first one will inherit security inconsistencies leading to an implementation that, as a consequence, will not enforce security requirements.

With respect to the process, phases 2, 4, and 6, are executed only if the diagrams they receive in input are consistent, in order to generate respectively consistent procedural models, security policies, and implementation skeletons.

B. Automated generation

The software tool fully automates the generation of procedural models, security policies and skeleton implementations, i.e., phases 2, 4, 6. The implementation is based on the transformation rules described in Section V, for phases 2 and 4, while for phase 6 we use a similar mechanism but, since the source and target languages are different, we used a different implementation solution, specified in [25].

Such transformations are the mechanisms that integrate the models and, since they are automated, they facilitate the generation of the implementation skeletons.

C. Procedural security policy verification

Procedural models, used to generate implementation skeletons, must satisfy security policies derived from the social/organizational security requirements. This is required for generating implementations that enforce social/organizational security requirements.

Few verification software engines for procedural models are available in literature, e.g., [1], [20]. But such approaches specify, and therefore verify, only the control flow of processes, while with SecBPMN2 procedural security policies, both the control flow and the message flows have to be verified.

For this, we created a verification engine based on $\mathcal{K}$ [9], which permits to check SecBPMN2-Q procedural security policies against SecBPMN-ml procedural models.

$\mathcal{K}$ is a logic-based planning language supported by $DVL^K$, an inference engine created to generate the set of actions to achieve a goal. We encoded business processes as a set of constraints in $\mathcal{K}$ so the plans that can be generated follow the control flow of the business process. We encoded the procedural security policies as $\mathcal{K}$ goals, that is, a desirable state of the world to be reached with the execution of a plan. Therefore, if $DVL^K$ generates a plan (a set of activities that follows a procedural model) that achieves a goal (a procedural security policy), the procedural model verifies the security policy.

Listing 2 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 first one represents any activity and the latter represents specific activities. Lines 1-3 define the message and the control flow fluents. The control flow fluent requires two activities as parameters and represents the control flow between two elements. The message flow fluents require as parameters two activities and a message, and represents the message flow between two activities. Lines 4-6 represent part of the procedural model in Fig. 3(a). In particular, the control flow between the generate transfer order and send transfer order and the message flow containing the message transfer order between the latter activity and analyse transfer order.

```
fluent:
1  controlFlow( T1,T2 ) requires activity( T1 ),
activity( T1 ),
2  messageFlow( T1,T2,M ) requires activity( T1 ),
activity( T1 ), message( M ).
4  initially:
5  controlFlow( generate transfer order, send transfer order ).
6  messageFlow( send transfer order, analyse transfer order, transfer order ).
```

Listing 2. A $\mathcal{K}$ representation of part of the process in Fig. 3(a)

Listing 3 shows the $\mathcal{K}$ goal generated from the procedural security policy in Fig. 3(b). Line 1 specifies that a $\mathcal{K}$ goal is defined. 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 the predicate Line 2 instruct the planner to find a plan in which the activity analyse transfer order is executed by bank src. Line 4 specifies that the plan will contain a message sent by any activity to analyse transfer order and transmitting the message transfer order. Line 5 instructs the planner to find a plan where path1 is executed. path1 is defined as a set of constraints (Lines 7-8), where path1_j becomes true after analyse transfer order is executed and path1 becomes true if accredit value is executed and path1_j is true. Line 6 specifies how many actions the generated plan shall contain maximum, we set this as the number of all SecBPMN2 elements in the business process. This parameter does not influence the generated plans, if it’s 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.
For the sake of space, we do not include the transformation rules between ScBPMN2 and \( \mathcal{K} \). Documentation on such rules can be found at [31].

If DVL\(^K\) generates at least a plan that respects the constraints defined in Listing 2, which achieves the goal defined in Listing 3, then the procedural model used to generate the constrains is verified against the procedural security policy used to generate the goal.

### IX. PRELIMINARY EMPIRICAL EVALUATIONS

We evaluated the proposed framework in two ways. First, we checked the actual interest practitioners have about such a framework. This was done through a series of interviews with industry experts. Second, we evaluated the framework conducting an empirical experiment with modelers.

#### A. Interviews

We interviewed industry experts to evaluate the interest of companies about the framework. The interviews consisted in a description of the framework and 10 questions. They were conducted through Voice over IP (VoIP) calls or telephone calls and lasted between 10 and 20 minutes. We interviewed 11 subjects from companies such as HP, Tecnalia, ATOS, Eurocontrol, and Thales group.

64% of the subjects declared to be security experts and to perform regularly systematic security analysis, while 36% of them have a superficial knowledge on security. The subjects interested in a future usage of the framework are 64%, in particular 86% of the security experts showed high interest, while only 14% of them declared not to be interested. The subjects not interested in the framework are 36% but, interestingly, 75% of them are not security experts and only 25% of them are. This data leads to the conclusion that there is a good interest among the subjects (87%) are experienced on goal-based modeling requirements engineers with experience in modeling socio/organizational and procedural models, to generate the latter from the former, and to modify the procedural model. Following Wohlin’s guidelines [33], the relevant validity threats for the experiment are: (i) low statistical power: the number of subjects can be low, since we invited only security requirements engineers with experience in modeling socio/organizational or procedural models; (ii) selection: the participation to the experiment was volunteer-based, therefore subjects might be enthusiastic. We mitigate this threat being as neutral as possible about the framework during the presentation.

#### B. Empirical experiment

We designed and conducted an empirical experiment to evaluate the effectiveness of the process supporting the framework, the automated transformations, and the usability of the supporting toolset. This evaluation is guided by the following research questions:

**RQ1: Does the process effectively help the users?** The criteria taken into account for evaluating the process are: (i) how much its execution helps security requirement engineers in specifying security requirements; (ii) how well it adapts with the software engineering process used by the subjects; (iii) how easy is to follow the process, when performing the exercises proposed in the experiment.

**RQ2: Do the automated transformations effectively help the users?** We evaluated how much automated transformations help in specifying business processes and security policies, how easy they are to use and if they can be used without using transformation rules. We directly asked to subjects about their feeling on automated transformation and we recorded how many transformations were performed during the exercises proposed in the experiment.

**RQ3: Is the software tool usable?** We evaluated the usability with the standard System Usability Scale (SUS) [30]. The original SUS [30] was used with a slight adaptation: for each item of the scale, the word system was replaced with tool, to ease the understanding of the scale. The scale for assessing the tool consisted of 10 items, like in the original SUS. Apart from the tool usability, we asked about how well the software tool supports the framework.

### C. Design of the experiment and threats to validity

The experiment was divided in five parts: (i) introduction of the framework and modeling languages; (ii) execution of small exercises; (iii) introduction to the software tool; (iv) execution of an exercise that requires following the process and using the automated transformations with the help of the tool; (v) final questionnaire.

The fourth part required the subjects to navigate between social/organizational and procedural models, to generate the latter from the former, and to modify the procedural model.

8 subjects participated to the empirical experiment. All subjects were Ph.D. students and expert modelers. Most of the subjects (87%) are experienced on goal-based modeling languages, and 38% are experienced business process modelers. 63% of the subjects has an average security knowledge, 25% are security experts, while 12% does not have any knowledge on security. The sample contains a heterogeneous representation of experienced modelers.

The exercises in the second part of the experiment were executed to assess if the subjects correctly understood the concepts introduced in the first part of the experiment. 57% of the subjects executed the exercises without errors, while 43% did only 1 error. No subjects did more than one error. This indicates that the baseline concepts, explained in the first
part of the experiment, were well understood; a key point to properly follow the hand-on section, execute the exercise in the fourth section and, therefore, obtain significant results.

**RQ1** To evaluate the process, subjects should have used it to define medium-to-complex scenarios, but this was not possible because of time constraints. Therefore, some of the questions did not receive enough positive or negative answers, because many subjects felt they did not have the opportunity to apply the process extensively. The significant results are about how much the process eases the generation of business processes: 63% of the subjects believe it eases the generation, 38% have no strong feeling, while no one disagreed with such fact.

**RQ2** The subjects appreciated the automated transformation. In particular, 75% of them agree on the fact that it helps in creating business process models and in enforcing security requirements, 25% do not have any opinion, while no one disagreed. When asked whether the automated transformations help security requirement engineers in case of large scenarios, the answers were even more positive: 88% agreed, 12% did not have any opinion, and 0% disagreed. This emphasises that subjects believe in the usefulness of automated transformation, especially in case of real-world scenarios with large complex models. Subjects’ comments confirm such conclusion, but they also highlight that it may be difficult to properly use the automated transformation, especially for users who do not have a previous knowledge on STS-ml and SecBPMN2.

**RQ3** For what concerns the evaluation of the software tool, 88% of the subjects believe that it well integrates the modeling languages, 12% have no strong feeling, while no one believes that the tool does not do a good job in integrating the different modeling languages. For what concern the support to the process, 75% of the subjects believe the software tool facilitates the application of the process, 25% do not have any strong opinion, while no one disagreed. Furthermore, 63% of the subjects believe the tool is sufficiently fast, 37% have no opinion and no one believes the software is too slow.

As an overall result, the calculated mean SUS score of the tool is 64.4 (n=8, SD=9.7). SUS scores are normalized and range from 0 to 100, with 0 for least usable systems and 100 for best usable systems. On average, systems achieve approximately a SUS score of 68 [30]. Thus, the SUS score of the tool is close to this average, indicating that the tool is neither among the worst systems nor among the best with regard to usability. When the descriptive adjectives of the study of Bangor, Kortum and Miller [2] are applied to the SUS score from this evaluation, the tool can be labelled as between “good” and “ok”. Consequently, the current usability status of the tool can be considered “acceptable”. However, there is room for improvement with regard to its usability.

**X. RELATED WORK**

In the years past, several approaches have been proposed to address the verification of requirements in business processes [4], [21], [34], [24], [15]. 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 procedural approaches, while highlighting the differences with our approach considering the corresponding phases.

**Modeling BPMN with security concepts for procedural specification.** Many graphical modeling languages extending BPMN [18] with security aspects have been proposed. Ad-hoc notations are used in SecureBPMN [4] proposed by Brucker et al to capture security and compliance requirements. Other extensions of BPMN also rely on security annotated business process modelling [21], [34], [24], [15] similarly to our approach. However, differently from existing approaches, ours allows the definition of custom security policies. Instead, existing approaches employ software engines which use models created with the respective languages to check a fixed set of hard coded security policies. Examples of such engines include [34], [26], [23].

**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) [8] allow to graphically define queries and check which business processes satisfy the queries. These two query languages are not based on BPMN, which makes their applicability and, most importantly, their learning process slower than that of, for example, SecBPMN-Q that is based on the well-known standard. 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 [22] proposes a language and a framework that checks if the code of the software diverges from specified behaviors (i.e., policies). These approaches have a main drawback: 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, for they require a substantial effort for formalizing 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.

**Sadiq et al.** [23] 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. [14] 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 BPEL instead of BPMN, and it is not focused on security, but rather on regulatory compliance.

Ghose and Koliadis [10] 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 [1], [20] 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.

XI. CONCLUSION

We have proposed an STS-based framework to assist security requirement engineers in specifying and enforcing social/organizational security requirements in socio-technical systems. An iterative and incremental process supports the modeling, transformation, and analysis activities necessary to capture and enforce security requirements via a skeleton implementation. We evaluated both of the framework through interviews with practitioners and empirical experiment with modelers. The results confirmed the effectiveness and usefulness of the framework in enforcing security requirements. However, the experiment involved only researchers. We intend to conduct further empirical experimentation with domain and security experts from industry.

The framework suffers from these limitations: (i) limited application range: the choice of specific languages (STS-ml, SecBPMN and RDL) reduces the possible adoption of the framework; (ii) limited cognitive scalability: this problem, inherited from STS-ml, prevents to easily model real-world scenarios, because of the dimensions of the resulting diagrams.

Future work considers: (i) extending the framework to generate verified procedural security policies in the implementation; (ii) reducing the scalability issue; and last, but not least (iii) conducting empirical evaluation with practitioners.

ACKNOWLEDGMENT

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

REFERENCES

[18] OMG. BPMN 2.0, Jan 2011.