Analysing the Efficacy of Security Policies in Cyber-Physical Socio-Technical Systems

Gabriele Lenzini, Sjouke Mauw, Samir Ouchani

CSC/Interdisciplinary Centre for Security, Reliability and Trust,
University of Luxembourg, Luxembourg
{gabriele.lenzini,sjouke.mauw,samir.ouchani}@uni.lu

Abstract. A crucial question for an ICT organization wishing to improve its security is whether a security policy together with physical access controls protects from socio-technical threats. We study this question formally. We model the information flow defined by what the organization's employees do (copy, move, and destroy information) and propose an algorithm that enforces a policy on the model, before checking against an adversary if a security requirement holds.

Keywords: Socio-Technical-Physical Systems, Modelling Security and Policies.

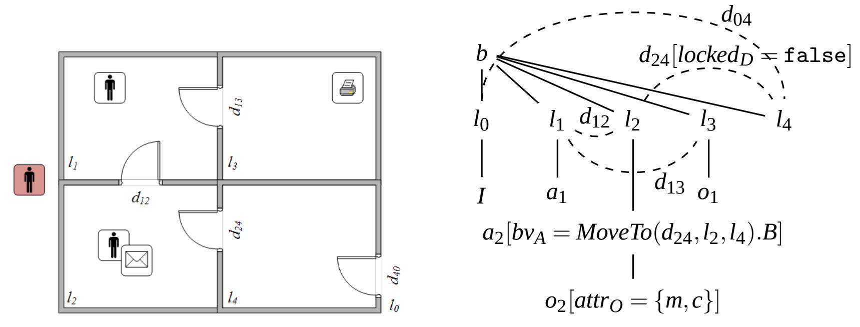
1 Introduction

The data-flow of an ICT organization is defined by what its employees do. They access, copy, share, and move pieces of information and objects that carry information, such as hard disks. In this flow, an organization must avoid to have critical data stolen. To reduce this risk, organizations protect the access to files, or use encryption. Paper documents or electronics are closed in drawers and offices are locked.

They also adopt policies, such as a "clean desk" policy, campaign for best practices, such as "always encrypt emails". However, a critical question remains whether a specific combination of policies and physical/digital controls is effective against certain threats. This question does not have an easy answer, but we advocate that it can be explored by the use of formal methods.

Formal methods have been successfully applied in the analysis of security protocols over the last decades (*e.g.*, see [1]). Recently they have also been proposed to model *Socio-Technical Physical Systems (STPS)* [2] —systems whose operation is defined by the interactions between people, technology, and physical elements— of which ICT organizations are examples. This new research (*e.g.*, see [3–7]) suggests that formal methods can be used in the analysis of the security of STPS and of the processes that define an STPS's daily work flow.

The idea of reasoning about a system's security in combination with policies has also been explored (see [8–12] and Section 3). Policies on accesses, on work flows, and on security properties have been formalized and verified over models of STPS. One work in particular, that of Hartel *et al.* [12] considers policies and requirements at the same time. Hartel *et al.* study four specific systems and model check, in SPIN, whether the systems composed with the policies comply with a given requirement.



(a) (b)

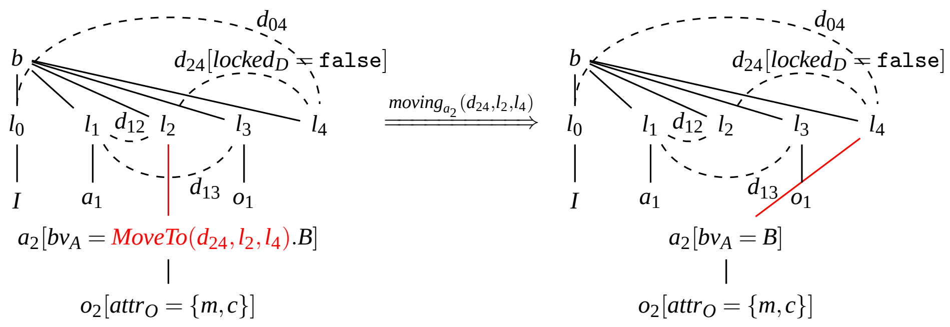
Fig. 1: (a) A simplified *STPS*'s state; (b) its formal representation.

Here, we aim to provide a formal framework to reason about the efficacy of a policy and of security control mechanisms against an adversary model. Let $\mathscr{S}$ be a model of an STPS, π be a model of a policy, p be a desirable security property, and I be an adversary model. The high-level formalization of the proposition whether $\mathscr{S}$ constrained by policy π is effective in realizing p can be expressed symbolically as follows: $\mathscr{S}_{|\pi} \models_I p$ (1). Appropriately instantiated, $\mathscr{S}_{|\pi}$ represents the (executions of) $\mathscr{S}$ where π is enforced, while $\models_I p$ is the relation "satisfies requirement p in the presence of adversary I". In absence of π, proposition (1) collapses into $\mathscr{S} \models_I p$, the classical proposition about whether $\mathscr{S}$ satisfies p in the presence of I. We develop a precise formal framework to express the abstract question in (1) and we develop an algorithm to compute $\mathscr{S}_{|\pi}$.

Background. There are a few formal languages upon which we can build. Here, we leverage on the one presented in [6]. There, the authors model an STPS's *state* as a labelled multi-graph. Nodes model offices, objects, or employees of the STPS. Edges represent either doors between offices or a (direct) location relation between nodes saying that "node x is contained/located in node y".

Figure 1 exemplifies an STPS's state and the graph representing it. Here (please, ignore the labels between '[ ]' for now) b is the building and $l_1 - l_4$ are its four rooms. Node l_0 models the outside. Node a_1, in room l_1, and node a_2, in room l_2, are employees. a_2 holds o_2, supposedly a letter. Node o_1, a printer, is in room l_3. I, the intruder, waits outside. Edges d_{04}, d_{12}, d_{13} and d_{24} are the doors between the four rooms.

Nodes and edges of the graph can be labelled to supply additional information. A label on an edge "door" expresses whether the door is locked and what key can lock/unlock it. A label on a node "object" tells us whether the object is movable (m), destroyable (d), or a container (c); in this case, another label tells us whether it is locked and what key can lock/unlock it. A label on a node "agent" tells us what is the "protocol" that defines the agent's behaviour. Agents can, for instance, move, pick up an object, put an object down, or destroy an object and its contents, or exchange an object with another agent. Figure 2 (left side) shows some of the labels (not all of them, though).



b
d_{04}
$d_{24}[locked_D = \texttt{false}]$
l_0
l_1
d_{12}
l_2
l_3
l_4
I
a_1
d_{13}
o_1
$a_2[bv_A = MoveTo(d_{24}, l_2, l_4).B]$
$o_2[attr_O = \{m, c\}]$
$moving_{a_2}(d_{24}, l_2, l_4)$
$a_2[bv_A = B]$

Fig. 2: How the *STPS* in Figure 1 changes because of agent a_2's moving to l_4.

The door d_{24} is unlocked; o_2 is a container and is movable; agent a_2 is about to move to room l_4.

Formally, an STPS's state is a tuple $\langle Phy, Obj, Act, E\rangle$. *Phy* is composed of nodes representing the physical spaces ($L \cup \{b\}$), door identifiers (D), and two door labelling functions ($locked_D$, and key_D); *Obj* is the nodes that model objects (O) and three object labelling functions ($attr_O$, $lock_O$, and key_O); *Act* is the nodes that model the employees and the intruder ($A \cup \{I\}$), and a set of labelling functions that return the protocols/behaviour of each employee (bv_A). Item E are the edges of the graph: the edges that model the doors ($L \times L$), labelled with door identifiers, and the edges that model the location of employees ($(A \cup \{I\}) \times L$) and of objects ($O \times (L \cup A \cup \{I\} \cup O)$).

The authors of [6] also describe how an STPS's state changes because of what the employees or the intruder do. The effect of an action is defined by a (conditional) graph-rewriting rule. A rule rewrites the formal graph representing the STPS's state by changing it as one intuitively expects: an agent's moving from one room to another, if the rooms are connected by an open door, has the effect of changing the agent's location to the new "room". For reasons of space we will not include the rewriting rules in this paper, but Figure 2 gives a rough idea of how the effect of the "move to" rule looks like. Note the transition (*i.e.*, arrow $\Rightarrow$) is labelled with the action that caused the transition.

Other rules define what the intruder I can do. He can be malicious, (*e.g.*, pick locks, steal or slip objects from people's pockets) but he cannot break the laws of physics: he cannot traverse walls, nor perform teleportation. I does not follow a protocol. All his actions are enabled, if they are possible.

Starting from a specific initial configuration, by applying the rules that are enabled one can generate *executions of the* STPS. The semantical model of these executions for a specific STPS is a *labelled transition system* $\mathscr{S} = \langle \mathbf{S}, \Gamma, S_0, \Rightarrow\rangle$ where $\mathbf{S}$ is the set of all possible STPS's states, Γ is the set of action labels, $S_0 \in \mathbf{S}$ is the initial state, and $\Rightarrow\ \subseteq\ (\mathbf{S} \times \Gamma \times \mathbf{S})$ is the labelled transition relation between states. It is the smallest relation that satisfies the graph rewriting rules. It must be stressed that $\mathscr{S}$ includes also the transitions due to the intruder.

In [6], $\mathscr{S}$ is a *probabilistic and weighted* labelled transition system. The agent's behaviour is probabilistic *i.e.*, specified by a stochastic process algebra. Actions have a weight, *i.e.*, cost. Costs are important in defining the intruder's strategy. For instance he can pick a lock, which may cost more (*e.g.*, in time) than opening the door with the key. He can also decide to use the key, but then the key must be retrieved (*e.g.*, stolen) first. All the details are in [6] but the intuitive description we have just given here is sufficient for understanding what we are proposing next.

2 Modelling Data and Data Flow

We extend [6] to be able to model data and the flow of data. For this purpose, we introduce *digital objects* and *digital object carriers*. A digital object models a piece of data, such as a file. Data cannot exist by their own: they need to be stored/carried. Digital object carriers are carrying data objects. Hard disks, USB pens, a book, (the mind of) an agent are data carriers. Formally, *Obj* is extended with a new labelling function $type_O$: returns p if the object is physical, d if the object is digital.

Digital objects can be cloned/copied, but they need a carrier that holds them afterwards. Formally, this means to extend *Act*, the language of an employee's or the Intruder's actions with two additional actions: $Clone(o,o')$ and $Clone(o)$. The former creates an identical copy of o into carrier o'; the latter "clones" o in the mind of who executes the action. All the other actions onto objects (*e.g.*, exchange them, destroy them, et cetera) remain applicable with the only constraint that digital objects need a carrier. Due to space limitations, we will not describe the new rules in this paper. The resulting formal semantics is a probabilistic and weighted labelled transition systems extending the one given in [6].

In the setting of this paper, we do not need probabilities or costs. We see little utility in policies that apply with certain probability/cost. Instead, in relation to (1), questioning whether a policy is effective to reduce the risk of a specific attack within a certain probability/cost is a legitimate question. In this paper we do not develop this probabilistic framework. We leave it for future work. Instead, we interpret (1) as the question whether a certain policy, when enforced, is effective in removing attacks (resp., ensuring security) altogether. A non-probabilistic non-weighted labelled transition system (which we still write $\mathscr{S}$ in the remainder of the paper) can be obtained from the model in [6] by substituting any probabilistic choice in all bv_A with a non-deterministic choice and by ignoring costs.

3 Security Policies and Security Requirements

According to the Cambridge Dictionary, a *security policy* is "a plan, or a document, specifying what to do in particular situations, and often how and when to do it". Policies, when enforced or followed, should have the effect of nudging specific practices to become compliant with its provisions.

Policies can be modelled in several ways (*e.g.*, as behavioural patterns [12], as first-order logic assertions [10], as markov decision processes [11]); here we model a policy focusing on the constraints it has on the executions of an STPS. We consider a *security policy* as a statement on what may never happen in the STPS execution. We abstract from the reasons why an STPS's executions appear to be constrained, disregarding how a policy is actually enforced (*e.g.*, access control systems, people having accepted the policy for ethical reasons or for fear of punishment): we assume it is enforced somehow. A *security requirement* instead is a desirable security property that we would like to be valid despite specific threats coming from an adversary. We model a security requirement as a classic security property [1].

We express policies and requirements using the language of *security statements*. It corresponds to Linear Temporal Logic (LTL) with 'Next' and 'Until' operators.

Definition 1. *A* security statement *is any expression in the language $L(\varphi)$, so defined:*

$$\varphi ::= \texttt{true} \mid \varphi_{SP} \mid \varphi \wedge \varphi' \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \; \mathrm{U} \; \varphi'$$

$$\varphi_{SP} ::= \varphi_{SP} \wedge \varphi'_{SP} \mid \neg\varphi_{SP} \mid d \in conn(l,l') \mid o \in key_D(d) \mid (x,a) \in Hist_D(d) \mid (x,a) \in Hist_O(o) \mid$$
$$z \in |type_O(o) \mid y \in attr_O(o) \mid loc_O(o) = l \mid o \in key_O(o') \mid o \in cont_O(o') \mid loc_A(a) = l \mid o \in cont_A(a)$$

Operators $\wedge$ and $\neg$ give the full power of propositional logic; operators $\bigcirc$ and U are sufficient to derive the other LTL operators, $\Diamond$ and $\Box$. Note that, $L(\varphi_{SP})$ is the sub-language of propositional logic expressions over the STPS's state. In the remainder, we indicate with φ any formula in $L(\varphi)$, and with φ_{SP} any formula in $L(\varphi_{SP})$.

The informal meaning of a state predicate in φ_{SP} can be guessed from the name of the statement. So, for instance, $o \in key_D(d)$ evaluates to true if and only if o is the key that closes/opens door d. The formal semantics is defined in term of $[\![\cdot]\!]_S$, the function returning the truth value of a security statement in a given state $S \in \mathbf{S}$:

$$[\![d \in conn(l,l')]\!]_S \text{ iff } (l,d,l') \in C$$
$$[\![(x,a) \in Hist_D(o)]\!]_S \text{ iff } (x,a) \in Hist_D(o)$$
$$[\![(x,a) \in Hist_O(o)]\!]_S \text{ iff } (x,a) \in Hist_O(o)$$
$$[\![y \in attr_O(o)]\!]_S \text{ iff } y \in attr_O(o)$$
$$[\![z \in type_O(o)]\!]_S \text{ iff } z \in type_O(o)$$
$$[\![loc_O(o) = l]\!]_S \text{ iff } (l,o) \in (E)^+$$
$$[\![loc_A(a) = l]\!]_S \text{ iff } (l,a) \in E$$
$$[\![o \in key_D(d)]\!]_S \text{ iff } o = key_D(d)$$
$$[\![o \in key_O(o')]\!]_S \text{ iff } o = key_O(o')$$
$$[\![o \in cont_O(o')]\!]_S \text{ iff } (o',o) \in (E)^+$$
$$[\![o \in cont_A(a)]\!]_S \text{ iff } (a,o) \in (E)^+$$

$Hist_D$ and $Hist_O$ keep the history of who has locked/encrypted a door or an object.

The semantics of φ is the standard semantics of an LTL formula (*e.g.*, see [13]). Assuming that $\texttt{Words}(\varphi) = \{\rho \in (2^{\varphi_{SP}})^\omega : \rho \models_I \varphi\}$ is the set of all ω-words (*i.e.*, possible infinite words) over the alphabet $2^{\varphi_{SP}}$ that satisfy φ, the satisfaction relation $\models_I \; \subseteq \; (2^{\varphi_{SP}})^\omega \times L(\varphi)$ is the smallest relation satisfying the following properties:

- $\rho \models_I \texttt{true}$
- $\rho \models_I \varphi_{SP}$ iff $[\![\varphi_{SP}]\!]_{\rho[0]}$
- $\rho \models_I \neg\varphi$ iff $\rho \not\models_I \varphi$
- $\rho \models_I \varphi_1 \wedge \varphi_2$ iff $\rho \models_I \varphi_1$ and $\rho \models_I \varphi_2$
- $\rho \models \bigcirc\varphi$ iff $\rho[1\ldots] \models_I \varphi$
- $\rho \models_I \varphi_1 \mathrm{U} \; \varphi_2$ iff $\exists j \geq 0 : \rho[j \cdots] \models_I \varphi_2$ and $\rho[i \cdots] \models_I \varphi_1, \; \forall 0 \leq i < j$

Here, for $\rho = S_0 S_1 \ldots \in (2^{\varphi_{SP}})^\omega$, $\rho[j \cdots] = S_j S_{j+1} \ldots$ is the suffix of ρ starting in the $(j+1)^{st}$ symbol S_j. Given $\mathscr{S} = \langle \mathbf{S}, \Gamma, S_0, \Rightarrow \rangle$, we say that a security statement φ is valid in $\mathscr{S}$, written $\mathscr{S} \models_I \varphi$, when $\texttt{Traces}(\mathscr{S}) \subseteq \texttt{Words}(\varphi)$, where $\texttt{Traces}(\mathscr{S})$ is the set of all *prefix closed traces* of $\mathscr{S}$ [13].

From the language of security statements we derive the language of security policies and of security requirements. Since we decided to model the effect of policies as constraints on the execution of an STPS, a security policy is a safety property or a negation of a liveness property. We may consider to extend this language in the future. Instead, we do not impose any restrictions on the language of security requirements.

Definition 2. *A* policy *is a security statement of the form* $\Box\neg\varphi_{SP}$ *or* $\neg\Box(\varphi_{SP} \rightarrow \Diamond\varphi_{SP})$.

Definition 3. *A* requirement *is a security statement.*

Example 1. The policy "o should be kept in l" is written as $(\Box\neg(\exists_{a\in A}.\{o\} \subseteq cont_A(a) \wedge loc_A(a) \neq l)$; the requirement "no one brings o outside" as: $\neg(\Diamond loc_O(o) = l_0)$.

4 Policy Constrained Semantics

According to Definition 2, when a policy is enforced, no execution of the system is expected to violate the policy. This should hold only when we consider executions that do not include actions of the intruder, because an intruder is, by definition, someone who is freed from playing by the rules. Such reflections lead to the following definitions:

Definition 4 (Honest Trace). *An* honest trace *is a trace whose underlying sequence of states,* $S_0 \cdot \ldots \cdot S_i \cdot S_{i+1} \cdot \ldots$ *is such that* $(S_i, S_{i+1}) \in \Rightarrow$*, for all* $i \geq 0$ *and where the label of* $\Rightarrow$ *is not the intruder's ID.*

We indicate the set of all honest traces of $\mathscr{S}$ as $\texttt{Traces}_H(\mathscr{S})$. Here, A is the set of honest agents. Another relevant set of traces for the framework is the set of traces that satisfies a given security statement.

Definition 5 (Trace satisfying φ). *A trace satisfying* φ *is a trace in* $\texttt{Traces}(\mathscr{S}) \cap \texttt{Words}(\varphi)$. *An honest trace satisfying* φ *is a trace in* $\texttt{Traces}_H(\mathscr{S}) \cap \texttt{Words}(\varphi)$.

We denote the set of all traces satisfying φ by $\texttt{traces}(\mathscr{S}, \varphi)$, and the set of all honest traces satisfying a φ by $\texttt{traces}_H(\mathscr{S}, \varphi)$.

We are now interested to distinguishing, in an execution without the intruder, the requirements whose validity can be changed if the policy is enforced from those whose validity is unchanged by it.

Definition 6 (Requirements/Policies Affectedness). *Let* φ *be a requirement,* π *a policy, and* $\mathscr{S}$ *be a model of execution of an STPS. We say that* φ *is affected by* π *in* $\mathscr{S}$*, and we write it* $\varphi \leftharpoondown \varphi'$*, when* $\texttt{traces}_H(\mathscr{S}, \varphi) \subseteq \texttt{traces}_H(\mathscr{S}, \neg\pi) \neq \emptyset$.

Property 1. Relation $\leftharpoondown$ is reflexive and commutative.

In a system where a policy is parsimoniously enforced, no requirement must change their validity, except those that are affected by the enforcement of π.

Definition 7. *Let* $\mathscr{S} = \langle \mathbf{S}, S_0, \Rightarrow\rangle$ *be an STPS,* π *a policy. The* system $\mathscr{S}$ constrained by π*, written* $\mathscr{S}_{|\pi}$*, is a new* $\mathscr{S}' = \langle \mathbf{S}', S_0, \Rightarrow'\rangle$ *that satisfies the following conditions:*

1. *If* $\mathscr{S} \not\models_H \pi$ *then* $\mathscr{S}' \models_H \pi$*;*
2. *For all* p *such that* $p \not\leftharpoondown \pi$*, if* $\mathscr{S} \models_H p$ *then* $\mathscr{S}' \models_H p$.

Algorithm 1 Reduce $(\mathscr{S}, \pi) \to \mathscr{S}_{|\pi}$

```
**Case 1:** $\pi = \Box\neg\varphi_{SP}$:
 **Forall** $S_i \in \mathbf{S} : \exists\rho \in \mathtt{Traces}_H(\mathscr{S})$,
  $\rho = S_0 \cdots S_i \cdots$ **and** $\rho[i \cdots] \models \varphi_{SP}$ **Do**
  $\mathbf{S}' := \mathbf{S} \backslash \{S\}$;
**Forall** $(S', S_i) \in \Rightarrow$ **Do** $\Rightarrow' := (\Rightarrow \backslash \{(S', S)\}) \cup \{(S', S')\}$;
**Forall** $(S_i, S') \in \Rightarrow$ **Do** $\Rightarrow' := (\Rightarrow \backslash \{(S', S_i)\})$.

**Case 2:** $\pi = \neg\Box\varphi_{SP} \to \Diamond\varphi_{SP}$:
 **Forall** $S_i, S_j \in \mathbf{S} : \exists\rho \in \mathtt{Traces}_H(\mathscr{S})$
  $\rho = S_0 \cdots S_i \cdots S_j \cdots$ **and** $\rho[i \cdots] \models \pi\}$ **Do**
  $\Rightarrow' := (\Rightarrow \backslash \{(S_{j-1}, S_j)\}) \cup \{(S_{j-1}, S_{j-1})\}$.
```

Definition 7 makes it clear that in a system where the policy is enforced the policy holds, and that the validity of properties that are not affected by the policy does not change. The use of the $\models_H$ notation in Definition 7 stresses that the policy is enforced on the system's execution without the interference of the intruder. The intruder can still find its way into breaching security even in the constrained system. Actually, the constrained system will be secure only when $\mathtt{Traces}(\mathscr{S}_{|\pi}) \subseteq \mathtt{Words}(p)$. This is eventually the meaning we intended to give to the proposition in (1).

From an operational point of view we are interested to obtain $\mathscr{S}_{|\pi}$ from $\mathscr{S}$. Algorithm 1, inputs a $\mathscr{S} = \langle \mathbf{S}, S_0, \Rightarrow \rangle$, a π such that $p \not\leftarrow \pi$ and returns a labelled transition system for $\mathscr{S}_{|\pi}$. The new transitions are labelled with ε, the null action.

Proposition 1 (Soundness). *$\mathscr{S}'$ = Reduce$(\mathscr{S}, \pi)$ is a system constrained by π.*

Proof. (sketch) $\mathscr{S}$ satisfies the two conditions in Definition 7. A valid, in $\mathscr{S}$, requirement p that is not affected by π, will not change its validity due to the operations (removing states, and adding loops) that Algorithm 1 implements onto $\mathscr{S}$.

Proposition 2. *Reduce$(\mathscr{S}, \pi)$ can be implemented with worst-case time complexity $O(|\mathbf{S}|^2 \cdot check(\pi))$. Here, $check(\pi)$ is the complexity of checking π.*

Proof. (sketch) Algorithm 1 is inefficient: the **Forall**s browse far more states than necessary. A more efficient way is to search for the Ss with minimal index in a trace satisfying **Forall**s' conditions. Each **Forall**'s has at most $O(|\mathbf{S}|^2)$ iterations.

5 Conclusion

To reduce the risk that sensitive data are leaked, an ICT organization should protect its files and any item that may contain those files, such as hard disk, books, and USB pens. This can be done by restricting the digital and the physical access to data but also by implementing security policies meant to be enforced on the employees daily job. The research we presented in this paper sets the foundations for reasoning about an organization's security when it enforces its policies. We propose a formal approach: we represent the data flow as it is defined by the daily operation of the employees of an organization in a formal language. Policies are simple formulas that we use to restrict the possible evolution of the system and based on which we check the validity of a security property in the presence of an adversary.

Our theoretical approach focussed on clearly defining the relevant concepts while postponing the design of efficient algorithms. We believe that it is possible to reduce the complexity of our algorithm and to optimize the generation of our *STPS* models. It is worth to mention that, even using our non-optimized algorithm, we managed to run proof-of-concept scenarios by using the Probabilistic Symbolic Model Checker (PRISM) model checker. For reasons of space we could not report on this experience in the current paper. In the future, we will report on our practical experiences in full detail. Another future research question concerns our policy language. We kept our policy language simple to be able to focus on the main concepts in our framework. We will study the expressiveness of our language and study extensions needed to manage real policies. We will also consider the introduction of probabilities and costs in our framework.

References

1. C. Cremers and S. Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.
2. G. Baxter and I. Sommerville. Socio-technical Systems: From Design Methods to Systems Engineering. *Interacting with Computers*, 23(1):4–17, 2011.
3. R. De Nicola, G.L. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
4. C. Meadows and D. Pavlovic. Formalizing Physical Security Procedures. In *Security and Trust Management*, volume 7783 of *LNCS*, pages 193–208. Springer, 2013.
5. T. Sommestad, M. Ekstedt, and H. Holm. The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures. *IEEE Systems Journal*, 7(3):363–373, 2013.
6. G. Lenzini, S. Mauw, and S. Ouchani. Security Analysis of Socio-Technical Physical Systems. *Computers and Electrical Engineering*, 47(C):258–274, 2015.
7. T. Dimkov, W. Pieters, and P. Hartel. Portunes: Representing Attack Scenarios Spanning through the Physical, Digital and Social Domain. In *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, volume 6186 of *LNCS*, pages 112–129. Springer, 2010.
8. Philip W.L. Fong. Relationship-based Access Control: Protection Model and Policy Language. In *The First ACM Conference on Data and Application Security and Privacy*, CODASPY ’11, pages 191–202, 2011.
9. M. Jaume. Semantic Comparison of Security Policies: From Access Control Policies to Flow Properties. In *2012 IEEE Symposium on Security and Privacy*, pages 60–67, 2012.
10. S. Ranise and R. Traverso. ALPS: An Action Language for Policy Specification and Automated Safety Analysis. In *Security and Trust Management*, volume 8743 of *LNCS*, pages 146–161. Springer, 2014.
11. M.C. Tschantz, A. Datta, and J.M. Wing. Formalizing and Enforcing Purpose Restrictions in Privacy Policies. In *2012 IEEE Symposium on Security and Privacy*, pages 176–190, 2012.
12. P. Hartel, P. V. Eck, S. Etalle, and R. Wieringa. *Modelling mobility aspects of security policies*, volume 3362 of *LNCS*, pages 172–191. Springer, 2004.
13. Ch. Baier and J-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.