Ensuring the Functional Correctness of IoT through Formal Modeling and Verification

Samir Ouchani

Ensuring the Functional Correctness of IoT through Formal Modeling & Verification

Samir Ouchani

LINEACT, Laboratoire d'Innovation Numérique;
École d'Ingénieur en Informatique, CESI eXia, Aix-en-Provence, France.

Abstract. Recent research initiatives dedicated to formal modeling, functional correctness and security analysis of IoT systems, are generally limited to, model abstract behavioral patterns and look forward possible attacks beneath gauging and providing feasible attacks. This research considers the complementary problem by looking for more accurate attacks in IoT by capturing richer behaviors -technical, physical, and social- including their quantitative features. We propose IoT-SEC framework that establishes an adequate semantics to the IoT's components and their interactions including social actors that behave differently than automated processes. For security analysis, we develop a general approach based on a library of attack trees from where we generate automatically the monitor, the security policies and requirements to harden the IoT model and to check how well the model is secure. We use PRISM model checker to analyze the functionality and to check security of the IoT model. Precisely this contribution ensures the functionality of IoT systems by analyzing their functional correctness.

Keywords: IoT; Security Assessment; Attack Tree; Security Policies; Formal Verification; Formal Modeling; Model Checking; Functional Correctness.

1 Introduction

Internet of Things (IoT) is the network of physical objects -devices, vehicles, buildings and other items embedded with electronics, software, sensors, and network connectivity- that enables to collect and exchange massively data. This technology of intelligent device-to-device communication provides the much-needed leverage to IoT which make it growing extensively. It promises immense potential for improving the quality of life, health-care, manufacturing, transportation, etc.From a technology perspective, the rise of IoT is not changing widely while using the same technology, connectivity, and trimmed mobile applications. In this context, the challenging issue is checking and ensuring functionality, security and privacy of IoT from the existing and hidden vulnerabilities of the linked objects and the expanded inefficient cyber-security. Behinds, many attack vectors are difficult to manage and to get protected from in IoT especially against computational, memory, and energy limitations due to the large amount of data and messages; e.g. insecure web, cloud, mobile interfaces, network services, and the lack of transport encryption, etc.

For example in IoT health-care system, objects are engaged to monitor remotely patients and in case of a substantial change in the critical data, a notification is sent to

alert emergencies. Objects such as fit-bits and pacemakers enclosing different sensors like EEG, BP, ECG, and EMG are deployed to control blood pressure, hearing, etc.For communication, IoT uses a wide range of protocols to transport real-time data which make it critical to ensure the integrity of data and its inaccessibility for unauthorized users. Further, in crisis situations, patients are generally weak which make them an easy target against social engineering attacks [10]. At this level of complexity, security analysis of IoT is tricky while the components of the game are of different nature: people, physical and digital objects, software, cloud services, and infrastructures of multiple forms. We strengthen our analysis methodology by relying to security protocols and formal methods [12,13] to handle different type of IoT assets, and their communications that may happen via conventional and non-conventional protocols (e.g. visual, auditory, kinesthetic). Despite the raising interest in this subject, we target to develop sound techniques that help to automate the security analysis of IoT and to scrutinize *whether*, *how*, *at what cost*, and *with which probability*, IoT is secure.

Contributions. This research, firstly, develops IoT-SEC framework that initiates a modeling formalism by capturing the underlying semantics of IoT which is flexible to be extended for more elaborated features. It is rich by covering social behaviors, physical and digital objects, communication protocols, internal and external servers, and computation and storing cloud services. The formalism proposes assigning a cost e.g. time, to the execution of atomic actions, and the IoT components may behave non-deterministically, probabilistically, or deterministically where actions can be guarded by contextual conditions. The formalism also models a library of intruders, as particular process proper to each IoT components, able to act maliciously according to realistic abilities and specific conditions.

Further, this research develops a security analysis methodology for IoT. It is a statistical analysis and model-checking based approach built-up over PRISM tool [9]. To automate their use, we define a mapping from IoT models, expressed in the proposed formalism, to PRISM. Further, to overcome the downside of the expressiveness of monitors and security properties used in PRISM, we propose a library of pre-configured attack trees and we develop instantiation mechanism that help to generate automatically relevant monitors and security properties. Unfortunately due to the limited space, we focus only on the modeling mechanism and the correctness validation approach.

Outline. In summary, we review the related work in Section 2 and we describe the main components and goals of the global framework in Section3. Then in Section 4, we develop a theory to model for IoT and we detail our approach focusing mainly on the functional correctness. In section 5, we develop a tool that shows the obtained experimental results. Finally, Section 6 concludes the paper and sketches the future directions.

2 Related Work

To position our contribution in literature, we compare it within the works that deal with modeling, functional analysis, and security specification, and protocols in IoT. Since IoT research is young, the recent initiatives survey the IoT issues and challenges.

A. Habtamu [1] discusses guidelines to how adapt security standards, practices, and technologies in IoT. Fink *et al.* [3] classify the vulnerabilities that might arise high impact in IoT. In fact, they discuss a specific class of threats without precising its applicability on which configurations. To trustworthy a model they propose to exploit the physical randomness in IoT to generate keys for authentication and access control that ensure anonymity, likability, and observability. Xu *et al.* [17] survey design and security challenges in IoT. They propose the digital physical un-clonable function as solution to enable the direct use of hardware security primitives inside an arbitrary digital logic to create secure information flow and public key protocols that require only one clock cycle. Zhang *et al.* [18] highlight the ongoing challenges in IoT,especially identification, authentication and authorization, privacy, protocols, the related systems and software vulnerabilities. We believe that our framework contributes very well to the discussed challenges and it is a strong starting point to develop and extend easily the discussed research directions.

Hu *et al.* [5] proposed a face identification and resolution based technique for fog computing to improve processing capacity and save the bandwidth in IoT. To check security and preserve privacy, they propose an authentication and session key agreement protocol using data encryption and integrity checking by expressing CIA attributes in BAN logic. Islam *et al.* [6] analyzes security requirements in the presence of threat models for a health care scenario by minimizing security risk. They rely on the existing e-health policies and regulations to determine how much a requirement is violated. Ould-Yahia *et al.* [15] apply Ant colony optimization to care-off between random and uncertain behavior of sensors deployed during medical diagnosis towards e-health measures for IoT and intelligent social insects. The differences between intensities of measures result on the affected or safe path of the propagation of medical information show and quantify different e-health security vulnerabilities. Mohsin *et al.* [11] proposed a security analysis approach based on SMT for IoT entities mainly device configurations, network topologies, user policies and their related attack surfaces. Entities are formulated as a high-order logic formula, and the policies are a set of discrete constraints. To check the existing vulnerabilities, SMT solver outputs the possible solutions satisfying the constraints within an attack formula. Compared to our framework, this one is applicable only to a well guided configuration and scenario. The proposed approach is limited to a strict IoT schemes and the analysis method is not automated.

F. Kammüller *et al.* [7,8] investigate how Isabelle might help to improve detection of attack traces in IoT e-health by combining ethical requirement elicitation with automated reasoning. To provide trustworthy and secure IoT for vulnerable users in health-care scenarios, they employ high level logical modeling using dedicated Isabelle frameworks for: infrastructures, human actors, security policies, attack tree analysis, and security protocol. Torjusen *et al.* [16] present the high level instantiation of the runtime verification in color Petri net and its validation. They integrate runtime verification enablers in the feedback adaptation loop to guarantee the achievement of self-adaptive security and privacy properties for an e-health settings. At run-time, they enable the contextual state model, the requirements specifications, and the dynamic context monitoring and adaptation.

With respect to the commented work, IoT-SEC covers the probability and costs of actions, formalizes IoT, analyzes the correctness and measures their security level. Moreover, IoT-SEC is automatic by relying on the probabilistic model checking and it takes advantage from the algorithms built within.

3 IoT-SEC Framework

Prior deeper details, we explore first the IoT architecture adopted in IoT-SEC framework, then we overview the global analysis approach and the proposed security model.

3.1 Architecture

We describe the IoT architecture by presenting its components and their interactions. Figure 1 illustrates the proposed IoT architecture enclosing five main components, object devices are physical objects embedded with sensors and software, user devices are physical objects that communicate with servers and collect data from objects, computing services provided by internal, external, and cloud servers; social actors are human agents that can hold and manipulate devices, the environment is the infrastructures and spaces that envelops the IoT entities.

These components interact through communication protocols of different ranges (Human-machine, Bluetooth, ZigBee, WiFi, Cellular, SSH, IpSec, etc.).

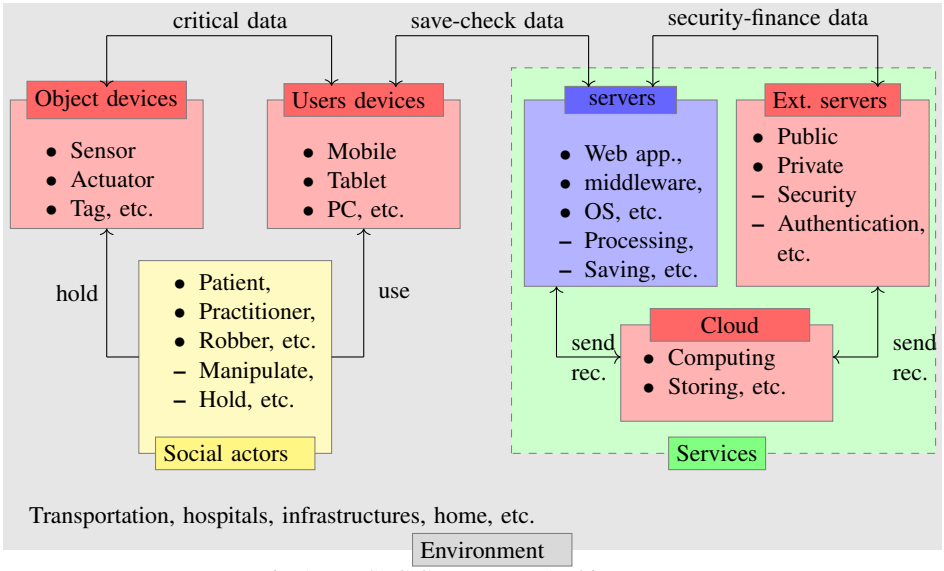


Fig. 1: IoT-SEC Components Architecture.

3.2 Methodology

The IoT methodology depicted in Figure 2 shows the main involved steps to evaluate and ensure the well functionality in IoT. It takes as input the IoT model M_{IoT}, the intruder model A_{IoT}, and a library of attack-trees T_{IoT}. First, an instantiation of A_{IoT} ($\widehat{A}_{IoT}$) is generated by the function $\mathscr{G}_A$ to contend M_{IoT} in order to produce a composed model $\widetilde{M}_{IoT}$. For security analysis the composed model $\widetilde{M}_{IoT}$ is abstracted then mapped into a PRISM code (M_P) by the function $\mathscr{T}_P$ [13].

The approach also demonstrates the use of T_{IoT} which produces relevant attack trees $\hat{T}_{IoT}$ to the composed model. To benefit from, the function $\mathscr{G}_{M,P}$ instantiates from $\hat{T}_{IoT}$ a temporal logic formula that expresses the security property and a monitor that control the mal-behaves of the intruder. Finally, the tool ($\models$) checks the satisfiability of the security properties in the considered model, and produces the verification result in terms of probability and cost.

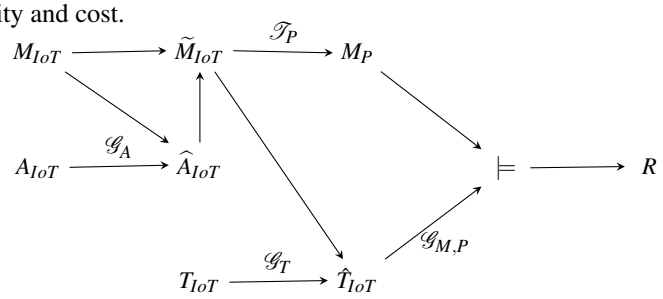


Fig. 2: IoT Methodology.

In the current work we focus only on ensuring the functional correctness instead of analyzing security.

4 Functional Correctness

To ensure the functional correctness [14] of an IoT-based system, we rely on IoT-SEC framework presented in Section 3 by extracting the approach depicted in Fig. 3 that shows the main steps to be followed in order to answer safely if the system under test functions properly or not, and/or with which probability/cost it can fail. We describe the steps as follows.

- IoT architecture defines the components composing an IoT-based system including social and non-social actors, sensors, applications, web services, physical infrastructures, etc.Further the way they communicate and interact.
- IoT model formalizes the architecture in a process algebra form by precizing the atomic actions for each component and the composition operator between each couple or group of components.
- IoT requirements express in PCTL formula different functional properties that we need to ensure.
- PRISM code is the transformation of the IoT model into the PRISM input language. This function should be an isomorphism i.e. each action defined in the IoT model has only one comportment that differs from the others.

– PRISM checks how much a requirement is ensured on the IoT model.
– Results are the output of PRISM, and it can be qualitative (true or false), or quantitative (a probability or a cost).

Following the above described steps we detail the modeling, the generation of PRISM code, and the expression of the requirements.

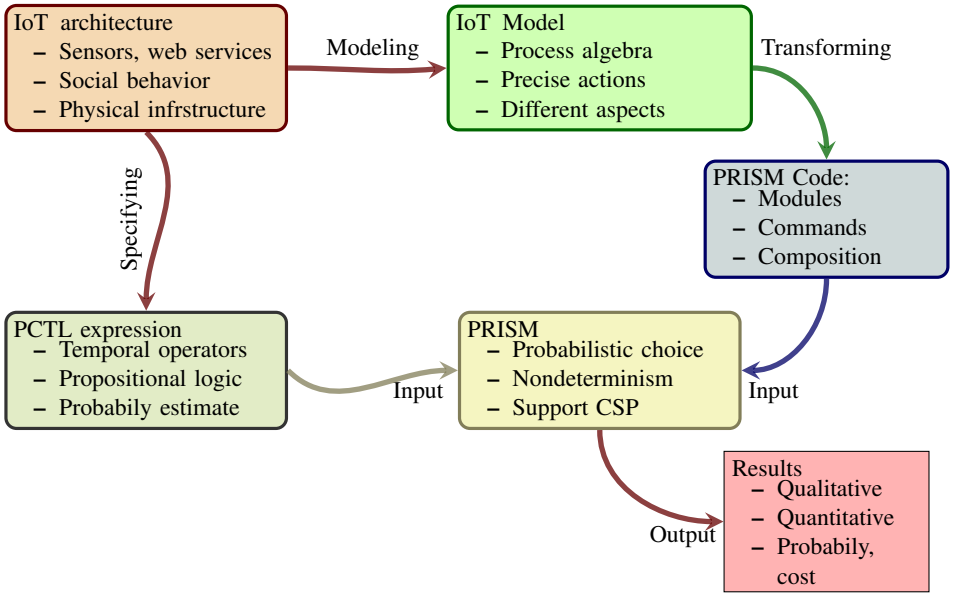


Fig. 3: Functional Correctness Framework for of IoT.

4.1 IoT Formal Model

Here we develop a formal model by considering the IoT architecture previously showed in Figure 1 as a composition of interconnected physical objects (devices and controllers, e.g. sensors and buildings), mobiles applications, cloud and computing online services, and people. We describe an IoT system S by the tuple $\langle Obj, Srv, Act, Env, Prot\rangle$ that defines formally the IoT entities: the connected objects (Obj), the environment (Env), the client-server applications and services (Srv), the social actors (Act), and the communication protocols ($Prot$) that ensure the interaction and the communication between the different types of IoT entities.

Objects An object can be either physical (e.g. sensor, USB key) or digital (e.g. data, message, information) with different specificities and abilities. An object can be a container, lockable (by digital or physical key), movable or/and destroyable by a program, an intelligent or human being actor. Sensor objects send data to the apps and receive it from the environment. An object Obj is a tuple $\langle O, attr_O, Actuator_O, \Sigma_O, Beh_O\rangle$, where:

- O is a finite set of tags $\varepsilon_o, o, o', o_i, \cdots \in O$ identifying the objects, and ε_o is the empty object.
- $attr_O : O \to 2^{\mathbb{T}}$ returns the attributes of an object, where $\mathbb{T} = \{p, c, m, d, r\}$, p stands for physical, c for container, m for movable, d for destroyable, and r for reproducible.
- $Actuator_O : O \to L \times 2^O \times O \times \mathbb{B}$ returns the tuple $\langle loc_O, cont_O, key_O, locked_O \rangle$ that specifies the status of an object o by specifying respectively its: location, contained objects, key, and if it is locked or not.
- Σ_O is a finite set of atomic actions that can be executed by an object, where:

$$\Sigma_O = \{\texttt{Start}_O, \texttt{Terminate}_O, \texttt{Send}_O(o,o'), \texttt{Receive}_O(o,o'), \texttt{Update}_O(o,o'), \texttt{Lock}_O(o,o'), \texttt{Unlock}_O(o,o'), \texttt{Move}_O(l,l') : o,o' \in O \text{ and } l,l' \in L\}$$

  $\texttt{Start}_O$ and $\texttt{Terminate}_O$ starts and terminates the process of an object, $\texttt{Send}_O(o,o')$ and $\texttt{Receive}_O(o,o')$ sends and receives o to/from o', $\texttt{Update}_O(o,o')$ updates o by o', $\texttt{Lock}_O(o,o')$ and $\texttt{Unlock}_O(o,o')$ lock and unlock o with o', respectively.
- $Beh_O : O \to \mathscr{L}_O$ returns the expression written in the language $\mathscr{L}_O$ that describes the behaviour of an object. The syntax of $\mathscr{L}_O$ is given by: $B_O ::= \texttt{Start}_O \cdot B_O \cdot \texttt{Terminate}_O \mid \alpha_O \cdot B \mid \alpha_O +_{g_o} \alpha'_O \mid \alpha_O$, where $\alpha_O \in \Sigma_O \backslash \{\texttt{Start}_O, \texttt{Terminate}_O\}$ and "$\cdot$" composes sequentially the actions, and $+_{g_o}$ is a guarded choice decision.

Services *Srv* ensures a client-server architecture including client applications, computation servers and web services. *Srv* is presented by the tuple $\langle V, O_V, srv_V, \Sigma_V, Beh_V \rangle$, where:

- V is a finite set of computing and storage services v, v', etc.
- O_V is a finite set of physical objects hosting services from V.
- $srv_V : O_V \to 2^V$ assigns for a given object a set of services.
- Σ_V is a finite set of actions supported by a service V , where:

$$\Sigma_V = \{\texttt{Start}_V, \texttt{Terminate}_V, \texttt{Send}_V(o,o'), \texttt{Receive}_V(o,o'), \texttt{Update}_V(o,o'), \texttt{Lock}_V(o,o'), \texttt{Unlock}_V(o,o') : o,o' \in O\}$$

  $\texttt{Start}_O$ and $\texttt{Terminate}_O$ starts and terminates the process of an object, $\texttt{Send}_O(o,o')$ and $\texttt{Receive}_O(o,o')$ sends and receives o to/from o', $\texttt{Update}_O(o,o')$ updates o by o', $\texttt{Lock}_O(o,o')$ and $\texttt{Unlock}_O(o,o')$ lock and unlock o with o', respectively.
- $Beh_V : O_V \to \mathscr{L}_V$ returns the behaviour of an object hosting a service. The syntax of $\mathscr{L}_V$ is expressed as follows: $B_V ::= \texttt{Start}_V \cdot B_V \mid \alpha_V +_{g_V} \alpha'_V \mid \alpha_V$, where $\alpha_V \in \Sigma_V \backslash \{\texttt{Start}_V\}$ and "$\cdot$" composes sequentially the actions and $+_{g_V}$ selects the left action if the guard g_V is true otherwise, the right action is selected.

Actors Actors are of different categories, they can be, patients hosting sensors, nurses, doctors, or any other types of agents. An actor interacts with others, manipulates objects, and accessing to resources by executing actions depends on his status and context. The execution is constrained by the environment, the possessed objects, the actor's intention and knowledge, and the access policies, etc. Formally, *Act* is a tuple $\langle A, categ_A, \Sigma_A, Bev_A \rangle$ where:

- A is a finite set of actors.
- $categ_A : A \rightarrow \mathbb{C}$ returns the category of an actor.
- $Actuator_A : A \rightarrow L \times 2^O$ returns the location ($loc_A \in L$) and the possessed objects ($poss_A \subseteq 2^O$) by an actor.
- The finite set of the actors actions Σ_A encloses all actions that can be executed by an agent.

$$\begin{aligned}\Sigma_A = \{&\texttt{Start}_A, \texttt{Moving}_A(l,l'), \texttt{Lock}_A(o,o'), \texttt{Unlock}_A(o,o'), \texttt{Send}_A(o,x),\\ &\texttt{Receive}_A(o,x), \texttt{Update}_A(o,o'), \texttt{Terminate}_A :\\ &l,l' \in L \text{ and } o,o' \in O \text{ and } a \in A \text{ and } x \in L \cup O \cup A\}\end{aligned}$$

  As the actions' names mean, they express respectively the moving between locations, locking/unlocking objects, sending/receiving objects from a location, an object, an actor; cloning or updating the content of an object (destroying and cloning objects are a special case of the update).
- $Bev_A : A \rightarrow \mathscr{L}_A$ returns the expression that describes the behaviour of an actor. It expresses the probabilistic decision and the cost (as time) of an execution. The syntax of $\mathscr{L}_A$ is generated by $B ::= Stop \mid \alpha_A.B \mid B + B \mid B +_g B \mid B +_p B$, where α is an atomic action in Σ_A, $+_p$ is a probabilistic decision, and $+_g$ is a deterministic choice.

Environment *Env* can be any human body or other natural species, or even a physical space that hosts objects to measure the needed metrics in order to be exploited/analyzed by the IoT system. In this model, we consider human body as an actor and the environment as a physical entity hosting all IoT entities. From this perspective we can model the environment as a connected container objects. Formally, *Env* is a tuple $\langle E, L, O_E, Actuator_E \rangle$, where:

- E is a finite set of environments denoted by e, e', etc..
- L is a finite set of locations (l, l', etc.).
- O_E is a finite set of physical objects of type container.
- $Actuator_E : O_E \times O_E \rightarrow 2^O$ returns the set of objects linking containers by physical objects (e.g. doors connecting two rooms).

Interaction Protocol *Prot* orchestrates and symphonies the communication and the interaction between the IoT entities. Since these entities differ in their nature, we define different communication protocols. Formally, *Prot* is a tuple $\langle Prot_{h,o}, Prot_{o,o}, Prot_{o,s} \rangle$ where $Prot_{h,o}$ ensures the communications between social actors and the objects, $Prot_{o,o}$ between objects, $Prot_{o,s}$ between objects and services on servers.

Considering an initial configuration of an IoT that defines the evaluation of objects, actors, and services attributes; *Prot* defines the changes of the attributes of each IoT entity regarding the executed actions. The IoT configuration is the association of all states of IoT entities and the changes of a configuration is ruled by transitions. An IoT's state $S = \langle S_O, S_V, S_A, S_E \rangle$ is composed from states of objects, services, actors, and the environment as an instance of $\langle Obj, Srv, Act, Env \rangle$. The transitions between states

are labeled and denoted by $S \overset{\ell,c,p}{\hookrightarrow} S'$, l names the action to be executed, c returns its cost and p is its probability value to be run. Due to the space limitation, we selected the following operational rules that synthesize transitions when two physical objects o and o' exchange a digital object o'' (SYN-O-O), an actor a takes an object o' from an object o (REC-A-O), and encrypt an object o' by an object o using o'' (LOC-O-O).

$$\frac{\begin{array}{c} Beh_O(o) = \texttt{Send}_O(o', [\![o'']\!]).Beh'_O(o) \wedge o'' \in cont_O(o) \wedge [\![o'']\!] \neq \varepsilon_o \\ Beh_O(o') = \texttt{Receive}_O(o''', [\![o'']\!]).Beh'_O(o') \wedge o''' \in cont_O(o) \wedge p \notin attr_O(o'') \end{array}}{\begin{array}{c} \langle\langle o, -, <-, \{o'', [\![o'']\!]\} >, -\rangle, \langle o', -, <-, \{o''', [\![o''']\!]\} >, -\rangle\rangle \overset{\texttt{Send}_O(o,o',[\![o'']\!]),c,p}{\hookrightarrow} \\ \langle\langle o, Beh'_O(o), <-, \{o'', [\![o'']\!]\} >, -\rangle, \langle o', Beh'_O(o'), <-, \{o''', [\![o''']\!]\} >, -\rangle\rangle \end{array}} \text{SYN-O-O}$$

$$\frac{\begin{array}{c} Bev_A(a) = \texttt{Receive}_A(o, o').Bev'_A(a) \wedge loc_A(a) = loc_O(o) \\ \neg locked_O(o) \wedge o' \in cont_O(o) \wedge p \in attr_O(o') \end{array}}{\begin{array}{c} \langle\langle a, -, <-, ->, -\rangle, \langle o, -, <-, \{o'\} >, -\rangle\rangle \overset{\texttt{Receive}_A(a,o,o'),c,p}{\hookrightarrow} \\ \langle\langle a, Bev'_A(a), <-, \{o'\} >, -\rangle, \langle o, Beh'_O(o), <-, ->, -\rangle\rangle \end{array}} \text{REC-A-O}$$

$$\frac{Beh_O(o) = \texttt{Lock}_O(o', o'').Beh'_O(o) \wedge \{o', o''\} \subset cont_O(o) \wedge [\![o', o'']\!] \neq \varepsilon_o}{\begin{array}{c} \langle\langle o, -, <-, \{o', o''\} >, -\rangle, \langle o', -, <-, ->, \neg locked_O(o')\rangle\rangle \overset{lock_O(o,o',o''),c,p}{\hookrightarrow} \\ \langle\langle o, Beh'_O(o), <-, \{o', o''\} >, -\rangle, \langle o', -, <-, ->, locked_O(o')\rangle\rangle \end{array}} \text{LOC-O-O}$$

We define an IoT's state and how this changes by the effect of actions as a labelled state transition system $\langle \mathbf{S}, S_0, \rightarrow \rangle$ where, $\mathbf{S}$ is the set of the IoT states, $S_0 \in \mathbf{S}$ is the initial state, and $\rightarrow \subseteq (\mathbf{S} \times \mathbf{L} \times \mathbf{S})$ the transition relation between states labled by $\mathbf{L}$. A transition $\hookrightarrow \in \rightarrow$ denoted by $S \overset{\ell,c,p}{\hookrightarrow} S'$ defines how IoT states change when the IoT entities behave. For example,

4.2 PRISM

PRISM is a probabilistic symbolic model checker that checks probabilistic specifications over probabilistic models. A specification can be expressed either in the probabilistic computation tree logic (PCTL) [2] or in a continuous stochastic logic. A model can be described using PRISM language. A PRISM program is a set of *modules*, each having a countable set of boolean or integer, local, variables. A module's state is fully defined by the evaluation of its local variables, while the program's state is defined by the evaluation of all variables, local and global.

In PRISM, the behavior of a module is defined by a set of probabilistic and/or Dirac commands that specifies textually the effect of an action in a probabilistic transition system. A probabilistic command is expressed by $[\alpha]\ g \rightarrow p_1 : u_1 + ... + p_m : u_m$, where p_i are probabilities ($p_i \in]0,1[$ and $\sum_{i=0}^{m} p_i = 1$), α is a label describing the name of an action, g is a propositional logic formula over local and global variables (i.e. a *guard*),

and u_i are *updates* for variables. An update, written as $(v'_j = val_j)\&\cdots\&(v'_k = val_k)$, assigns only values val_i to local variables v_i. It means that for a given action α, if the guard g is true, an update u_i is enabled with a probability p_i. The guard is an expression consisting of the evaluation of both local and global variables, and the propositional logic operators. The Dirac case where $p = 1$ is a command written simply by $[a]\ g \rightarrow u$.

Syntactically, a module named M is delimited by two keywords: the module head "`module` M", and the module termination "`endmodule`". Further, we can model costs with reward module R delimited by keywords "`rewards` R" and "`endrewards`". It is composed from a *state reward* or a *transition reward*. A state reward associates a cost (reward) of value r to any state satisfying g that is expressed by $g : r$. A transition reward has the form $[a]\ g : r$ expresses that the transitions labeled a, from states satisfying g, are acquiring the reward of value r.

PRISM supports also composition where modules communicate à la CSP process algebra (e.g. see [4]). For two modules M_1 and M_2, the following composition operators are supported.

- Synchronization: the full synchronization on all shared action is written as $M_1||M_2$,
- Interfacing: the parallel interface synchronization limited to the set of shared actions $\{a,b,\cdots\}$ is given by $M_1|[a,b,\cdots]|M_2$,
- Interleaving: the interleaving is expressed by $M_1|||M_2$,
- Hiding: $M/\{a,b,\cdots\}$ expresses hiding the actions $a,b,\cdots$ in the module M.
- Renaming: $M\{a \leftarrow b, c \leftarrow d,\ldots\}$ is to rename actions a by b, c by d, ...

4.3 Transformation of IoT to PRISM

To generate a PRISM program $\mathscr{P}$ proper to the provided IoT formalism, we define the function $\mathscr{T}_P$ that assigns for each IoT entity behavior its proper PRISM code fragment that is bounded by '`module IoT entity name`' and '`endmodule`' and the semantic rules of each action is expressed by a PRISM command.

Due to the space limitation, we present the PRISM commands of actions that their semantics rules are already defined in Section 4.1. The left side specifies the premises of a rule whereas the right side describes the results of the rules. For example, o_{o_2} is an atomic proposition showing the the object o possess o_2, l_a and l_o present the locations, and p_{o_3} precises the physicality attribute of o_3. Variables and propositions are evaluated first to describe the initial state of the IoT entities by relying on the tuple obtained by the *Actuator* proper to each entity.

$$\mathscr{T}_P(\alpha) = \begin{cases} [Syn_{o_2}]o_{o_2} \wedge o_{1_{o_3}} \wedge \neg p_{o_2} \wedge \neg p_{o_3} \rightarrow (o'_2 = o_2); & \texttt{iff:} \\ [Syn_{o_2}]o_{o_2} \wedge o_{1_{o_3}} \wedge \neg p_{o_2} \wedge \neg p_{o_3} \rightarrow (o'_3 = o_2); & \texttt{Send}_O(o_1,o_2) \in \Sigma_O^{o_1}, \\ & \texttt{Receive}_O(o_3,o_2) \in \Sigma_O^{o_2}. \\ [Tak_{o_1}]l_a = l_o \wedge o_{o_2} \wedge \neg lock_o \wedge p_{o_2} \rightarrow (a'_{o_2} = \top); & \\ [Tak_{o_1}]l_a = l_o \wedge o_{o_2} \wedge \neg lock_o \wedge p_{o_2} \rightarrow (o'_{o_2} = \bot); & \texttt{Receive}_A(o,o_2) \in \Sigma_A^a. \\ [loc_{o_1}]o_{o_1} \wedge o_{o_2} \wedge \neg k_{o_1} \wedge p_{o_1} = p_{o_2} \rightarrow (k'_{o_1} = \top); & \texttt{Lock}_O(o_1,o_2) \in \Sigma_O^o. \\ [loc_{o_1}]o_{o_1} \wedge o_{o_2} \wedge \neg k_{o_1} \wedge p_{o_1} = p_{o_2} \rightarrow (o'_{o_1} = \top); & \end{cases}$$

4.4 Functional Requirements

We comment here what properties can be of relevance and how to express them in such a way that they can be checked by running PRISM. A formalism that is able to express all the factors that diagrams describe, paths of actions, propositions on state variables, probabilities of occurrence of one or a sequence of actions.

PCTL formulas ϕ in such a logic are generated by the following BNF grammar:

$$\phi ::= \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{P}_{\bowtie p}[\psi] \mid \mathrm{R}_{\bowtie r}[F\phi]$$
$$\psi ::= \mathrm{X}\phi \mid \phi \mathrm{U} \phi \mid \phi \mathrm{U}^{\leq k} \phi$$

Here, $k \in \mathbb{N}$, $r \in \mathbb{R}^+$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. A state formula can be "ap", an atomic proposition, or any propositional expression built from "ap". $\mathrm{P}_{\bowtie p}[\psi]$, called *probabilistic path predicate*, returns true if the probability to satisfy the *path formula* ψ is $\bowtie p$. The *cost predicate* $\mathrm{R}_{\bowtie r}[\phi]$ returns true if the cost to satisfy ϕ is $\bowtie r$. Here, F is the temporal logic operator *eventually*. A path formula is built from the typical temporal operators *next* (X), *until* (U), and *bounded until* ($\mathrm{U}^{\leq k}$).

As usual, other logic operators can be derived from the basic operators, such as G refers to *Generally*. The semantics of these operators are given as follows.

- $\bot \equiv \neg\top$, $\phi \vee \phi' \equiv \neg(\neg\phi \wedge \neg\phi')$, $\phi \rightarrow \phi' \equiv \neg\phi \vee \phi'$, and
- $\phi \leftrightarrow \phi' \equiv \phi \rightarrow \phi' \ \wedge \ \phi' \rightarrow \phi$.
- $F\phi \equiv \ \top \ U \ \phi$, $F^{\leq k}\phi \equiv \ \top \ U^{\leq k} \ \phi$, $G\phi \equiv \ \neg(F\neg\phi)$, and
- $G^{\leq k}\phi \equiv \ \neg(F^{\leq k}\neg\phi)$ where $k \in \mathbb{N}$.
- $\mathrm{P}_{\geq p}[G\phi] \equiv \mathrm{P}_{\leq 1-p}[F\neg\phi]$.

Besides, Pmin, Pmax, Rmin, and Rmax are operators that can be used within path or state formulas to specify the minimum (resp. maximum) probability or cost.

5 Experiments Results

Here we apply the approach presented in Section 4, by following the discussed steps above, on a use case presenting a smart health care emergency room.

The IoT Architecture. Fig. 4 depicts the main components of a smart emergency composed of: one patient, two rooms, set of sensors, local server, and a station. The goal is to ensure a collection of defined functional requirements.

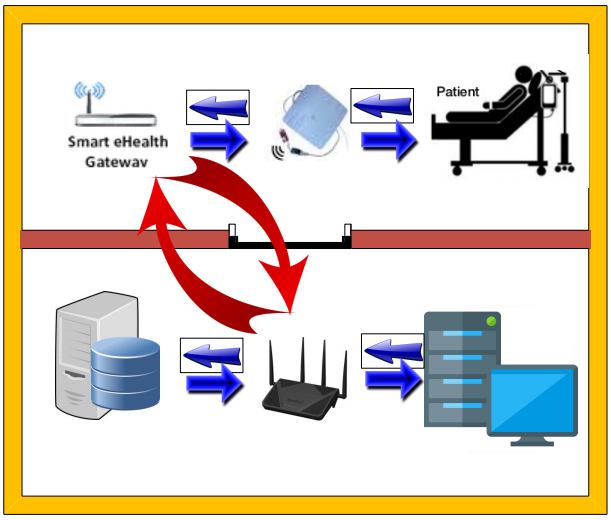


Fig. 4: smart emergency room

The IoT model. In the smart emergency presented in Fig. 4, two rooms l_1 and l_2 are accessible through the object o_1 (unique door) that is initially locked with the physical key o_1^k. The patient a_1 is in l_1 without possessing o_1^k but he has the sensor object o_1^s to measure his vital parameters and communicate it to the local server via the station o_1^d situated in l_2 at the end of medical services: monitoring, analysis, and cloud storage. Herein, we describe briefly the behaviours of the patient a_1, the sensor object o_1^s, the door o_1, the physical key o_1^k, and the station o_1^d, respectively.

- With a probability value of 0.3, a_1 can unlock o_1 before moving to l_2.
  $Bev_A(a_1) = \texttt{Start}_A.(\texttt{Unlock}_A(o_1,o_1^k) +_{0.3} \texttt{Moving}_A(l_1,l_1)).\texttt{Moving}_A(l_1,l_2).$
  $\texttt{Terminate}_A$ s.t. $Actuator_A(a_1) = \langle l_1, \{o_1^s\}\rangle$.
- o_1^k moves within its possessor, this possession is described with the guard g_1^k.
  $Beh_O(o_1^k) = \texttt{Start}_O.(\texttt{Move}_O(l_1,l_2) +_{g_1^k} \texttt{Move}_O(l_1,l_1)).\texttt{Terminate}_O$
  s.t. $Actuator_O(o_1^k) = \langle l_1, \varepsilon_o, \varepsilon_o, \perp\rangle$.
- o_1^s moves within a_1, and sends the value $[\![o_1^m]\!]$ received from a_1 to the station o_1^d.
  $Beh_O(o_1^s) = \texttt{Start}_O.((\texttt{Receive}_O(a_1,[\![o_1^m]\!]).\texttt{Update}_O(o_1^m,[\![o_1^m]\!]).\texttt{Send}_O(o_1^d,[\![o_1^m]\!]))$
  $+ (\texttt{Receive}_O(o_1^d,[\![o_2^m]\!]).\texttt{Update}_O(o_2^m,[\![o_2^m]\!])) + (\texttt{Move}_O(l_1,l_2)$
  $+_{g_1^s} \texttt{Move}_O(l_1,l_1))).\texttt{Terminate}_O$ s.t. $Actuator_O(o_1^s) = \langle l_1, \varepsilon_o, \varepsilon_o, \perp\rangle$.
- o_1^d synchronizes with o_1^s to send $[\![o_2^m]\!]$ and to receive $[\![o_2^m]\!]$.
  $Beh_O(o_1^d) = \texttt{Start}_O.((\texttt{Receive}_O(o_1^s,[\![o_1^m]\!]).\texttt{Update}_O(o_2^m,[\![o_1^m]\!])))$
  $+ (\texttt{Send}_O(o_1^s,[\![o_2^m]\!]))).\texttt{Terminate}_O$ s.t. $Actuator_O(o_1^d) = \langle l_2, \varepsilon_o, \varepsilon_o, \perp\rangle$.

The PRISM Model. For the performance assessment of the smart emergency, its IoT model is encoded into PRISM presented in Listing 1.1. It shows the code fragments of a_1, o_1^s, o_1^k, and o_1^d. Here we sketch a selected commands for each entity. The

module a_1 describes the behavior of a_1, its location l_{a_1} is initialized to the first room and its action $\texttt{Moving}_A(l_1,l_1)$ is expressed by the command M_{11}. The action $\mathrm{R}a_1(o_1^m)$ evaluates the body measure o_1^m. The status of o_1 is defined nondeterministically with actions U_{o_1} and L_{o_1} to evaluate equally the predicate $lock_{o_1}$. Actions in the module o_1^k assigns the locations of a_1 when it is possessed by him otherwise its location does not change. Further, o_1^s synchronizes with a_1 in $\mathrm{R}a_1(o_1^m)$ and with o_1^d in $\mathrm{S}o_1^s$ to receive $a_{o_1^k}$ sent by a_1. The module 'cost' assigns a cost of value 2 to the actions $\mathrm{R}a_1(o_1^m)$ and $\mathrm{S}o_1^s$. Furthermore, to add more entities, a user should just instantiates the proper module by renaming only its local variables.

```
mdp

module a_1
l_{a_1}: [1..2] init 1;
a_{o_1^s}: bool init true;
a_1(o_1^m): [1..5] init 1;
a_{o_1^k}: bool init true;
a_{Uo_1^k}: bool init false;

[U_{o_1}] (l_{a_1}=1)&(lock_{o_1})⇒
 0.3: (a'_{Uo_1^k}=true)+0.7:(l'_{a_1}=1);
[M_11](l_{a_1}=1)&(lock_{o_1})⇒(l'_{a_1}=1);
[M_12](l_{a_1}=1)&(¬lock_{o_1})⇒(l'_{a_1}=2);
[M_21](l_{a_1}=2)&(¬lock_{o_1})⇒(l'_{a_1}=1);
[M_22](l_{a_1}=2)⇒(l'_{a_1}=2);
[U_{o_1}](lock_{o_1})&(a_{o_1^k})⇒(l'_a=l_a);
[L_{o_1}](¬(lock_{o_1}))&(a_{o_1^k})⇒(l'_a=l_a);
[Ra_1(o_1^m)](a_1(o_1^m)<5)⇒
                (a_1(o_1^m)'=a_1(o_1^m)+1);
[Ra_1(o_1^m)](a_1(o_1^m)=5)⇒(a_1(o_1^m)'=1);
endmodule

module o_1
lock_{o_1}:bool init true;

[U_{o_1}](lock_{o_1}) ⇒ (lock'_{o_1}=false);
[L_{o_1}](¬(lock_{o_1})) ⇒lock'_{o_1}=true);
endmodule

module o_1^k
l_{o_1^k} : [1..2] init 1;

[M_11](a_{o_1^k})⇒ (l'_{o_1^k}=l_{a_1});
[M_12](a_{o_1^k})⇒ (l'_{o_1^k}=l_{a_1});
[M_21](a_{o_1^k})⇒ (l'_{o_1^k}=l_{a_1});
[M_22](a_{o_1^k})⇒ (l'_{o_1^k}=l_{a_1});
[M_22](a_{o_1^k})⇒ (l'_{o_1^k}=l_{a_1});

[](¬(a_{o_1^k}))⇒(l'_{o_1^k}=l_{o_1^k});
endmodule

module o_1^s
l_{o_1^s}:[1..2] init 1;
o_1^s(o_1^m):[0..5] init 0;

[M_11](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_12](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_21](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_22](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_22](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[Ra_1(o_1^m)](a_{o_1^s})⇒(o_1^s(o_1^m)'=a_1(o_1^m));
[So_1^s](o_1^m)!=0⇒ (o_1^s(o_1^m)'=a_1(o_1^m));
endmodule

module o_1^d
l_{o_1^s}:[1..2] init 1;
o_1^d(o_1^m):[0..5] init 0;

[M_11](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_12](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_21](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_22](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[M_22](a_{o_1^s})⇒(l'_{o_1^s}=l_{a_1});
[So_1^s](o_1^m!=0)⇒(o_1^d(o_1^m)'=o_1^s(o_1^m));
endmodule

rewards cost
true:1;

[Ra_1(o_1^m)] (l_a=2) : 2;
```

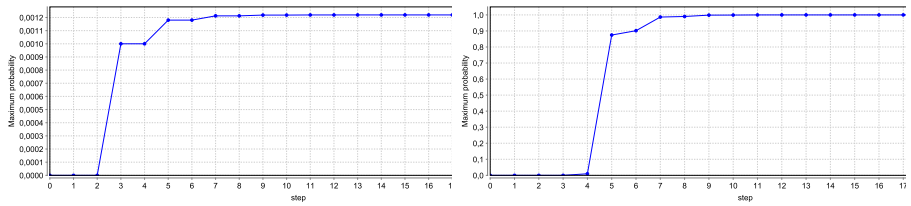
```
[So_1^s] (l_a = 2) : 2;
[] (a_1(o_1^m) > 3): 3;
[] (a_1(o_1^m) < 4): 2;
endrewards
```

Listing 1.1: The PRISM Fragment Code of the Smart Emergency.

The Functional Requirements. To ensure the functionality of the smart emergency system, we specify the following functional requirements.

1. *Property 1.* "What is the maximum probability for the patient a_1 to move from l_1 to l_2 when the measure of $a_1(o_1^m)$ is greater then 2?". The PCTL expression of this property is: $Pmax=?[(l_{o_1}=l_1)\wedge(a_1(o_1^m)<4)\ U \leq step\ (l_{o_1}=l_2)\wedge(a_1(o_1^m)>3)]$. The variable $step$ is the number of steps (transitions) to reach the state that satisfies: $(l_{o_1}=l_2)\wedge(a_1(o_1^m)>3)$.
2. *Property 2.* "What is the maximum probability to keep both the sensor object o_1^s and the station object o_1^d functioning together?". Its PCTL expression is: $Pmax=?[G(o_1^s(o_1^m)>0\wedge o_1^d(o_1^m)>0)]$.
3. *Property 3.* it looks to measure the minimum cost to read $a_1(o_1^m)$ and communicate it between o_1^s and o_1^d. It is expressed in PCTL by $Rmin=?[F(a_1(o_1^m)>0)]$.
4. *Property 4.* It measures the maximum cost for a_1 to move safely and keeping o_1^s functioning. Its PCTL expression is: $Rmax=?[F(o_1^s(o_1^m)>0)\{l_{a_1}=l_1, l_{a_1}=l_2\}]$.

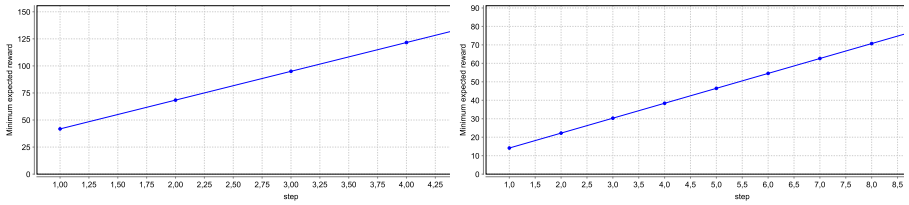
The Correctness Checking. The verification results of the above properties are depicted in Figure 5. The results of *Property 1* in Figure 5(a) show the convergence of the probability evaluation from 0 to 0.001 after 3 steps, then it increases up to 0.00125 after 9 steps. This result shows that the risk is low for a patient to move. Figure 5(b) shows that the probability obtained from the satisfiability of *Property 2* is 1 after step 6 and it converges to 0.9 after 4 steps. It means that the smart emergency model reliable at the most time.



(a) *Property 1.* (b) *Property 2.*

Fig. 5: The correctness checking results

The verification results depicted in Figure 6(a) show that the minimum reward value obtained from the satisfiability of *Property 3* is 121.59 and Figure 6(b) presents that the the cost to satisfy *Property 4* is at least 14.13. It means that the cost to keep the system always reliable is relatively high for communication and relatively low for the reliability of the smart emergency.



(a) *Property 3.* (b) *Property 4.*

Fig. 6: The correctness checking results

6 Conclusion

This paper sets the fundamentals of a fully automatic framework for modeling and analysis of IoT. Principally, we detail a part of it by presenting a formalism that captures the main structure and comportment of IoT entities covering physical and information infrastructures, services, assets, social actors, and also their activities and interactions. The execution of an action has a cost and guided by probabilities and/or contextual conditions. Further, the formalism has a rich and flexible semantics, which we use it to capture the IoT functional requirements expressing the possibility, the likelihood, and the cost of actions. Further, it is developed to be easy for other extensions and refinements. To carry our functional correctness analysis automatically, we devised an algorithm that maps an IoT model into the input language of PRISM in order to be checked against the requirements expressed in PCTL. Finally, the effectiveness of the proposed framework is validated on a case study.

This work sets the stage for further development. In the extended version of this work, we provide the complete set of rules, a detailed transformation function, and more experiments. Further, we intend to enrich our model with more assets: refine the contextual conditions, provide the security aspect of the IoT model, complete the other parts of the framework. Also from a solid theoretical point of view, we have to prove the correctness and the soundness of each developed step in a proof assistant (e.g. Coq). Furthermore, we implement the framework as a full standing tool and validated it on different case studies and real systems.

References

1. Habtamu Abie. *Adaptive Security for the Internet of Things: Research, Standards, and Practices*. Syngress Publishing, 1st edition, 2017.
2. Christel Baier and Joost Pieter Katoen. *Principles of Model Checking*. The MIT Press, may 2008.
3. G. A. Fink, D. V. Zarzhitsky, T. E. Carroll, and E. D. Farquhar. Security and privacy grand challenges for the internet of things. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pages 27–34, June 2015.
4. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, Incorporated, 1985.

5. P. Hu, H. Ning, T. Qiu, H. Song, Y. Wang, and X. Yao. Security and Privacy Preservation Scheme of Face Identification and Resolution Framework Using Fog Computing in Internet of Things. *IEEE Internet of Things Journal*, 4(5):1143–1155, 2017.
6. S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. S. Kwak. The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708, 2015.
7. F. Kammüller, J. C. Augusto, and S. Jones. Security and privacy requirements engineering for human centric iot systems using efriend and isabelle. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 401–406, June 2017.
8. Florian Kammüller. *Formal Modeling and Analysis with Humans in Infrastructures for IoT Health Care Systems*, pages 339–352. Springer International Publishing, 2017.
9. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer Verlag, 2011.
10. Gabriele Lenzini, Sjouke Mauw, and Samir Ouchani. Security Analysis of Socio-technical Physical Systems. *Comput. Electr. Eng.*, 47:258–274, 2015.
11. M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 180–188, Oct 2016.
12. S. Ouchani, O. A. Mohamed, and M. Debbabi. a security risk assessment framework for sysml activity diagrams. In *2013 IEEE 7th International Conference on Software Security and Reliability*.
13. Samir Ouchani, Otmane Ait Mohamed, and Mourad Debbabi. Efficient Probabilistic Abstraction for SysML Activity Diagrams. In *Proceedings of the 10th International Conference on Software Engineering and Formal Methods*, SEFM'12, pages 263–277. Springer-Verlag, 2012.
14. Samir Ouchani, Otmane Ait Mohamed, Mourad Debbabi, and Makan Pourzandi. *Verification of the Correctness in Composed UML Behavioural Diagrams*, pages 163–177. Springer Berlin Heidelberg, 2010.
15. Youcef Ould-Yahia, Soumya Banerjee, Samia Bouzefrane, and Hanifa Boucheneb. *Exploring Formal Strategy Framework for the Security in IoT towards e-Health Context using Computational Intelligence*, pages 63–90. Springer International Publishing, 2017.
16. Arild B. Torjusen, Habtamu Abie, Ebenezer Paintsil, Denis Trcek, and AAsmund Skomedal. Towards run-time verification of adaptive security for iot in ehealth. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 4:1–4:8. ACM, 2014.
17. Teng Xu, James B. Wendt, and Miodrag Potkonjak. Security of iot systems: Design challenges and opportunities. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, pages 417–423. IEEE Press, 2014.
18. Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh. Iot security: Ongoing challenges and research opportunities. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 230–234, Nov 2014.