A Formal Framework to Integrate Timed Security Rules within a TEFSM-Based System Specification

Wissam Mallouli*, Amel Mammar† and Ana Cavalli‡
*Montimage EURL, 39 rue Bobillot, 75013, Paris, France
Email: wissam.mallouli@montimage.com
†Institut Telecom SudParis, CNRS/SAMOVAR, Evry, France
Email: amel.mammar@it-sudparis.eu
‡Institut Telecom SudParis, CNRS/SAMOVAR, Evry, France
Email: ana.cavalli@it-sudparis.eu

***Abstract*—Formal methods are very useful in software industry and are becoming of paramount importance in practical engineering techniques. They involve the design and the modeling of various system aspects expressed usually through different paradigms. In this paper, we propose to combine two modeling formalisms in order to express both functional and security timed requirements of a system. First, the system behavior is specified based on its functional requirements using TEFSM (Timed Extended Finite State Machine) formalism. Second, this model is augmented by applying a set of dedicated algorithms to integrate timed security requirements specified in Nomad language. This language is well adapted to express security properties such as permissions, prohibitions and obligations with time considerations. The resulting secure model can be used for several purposes such as code generation, specification correctness proof, model checking or automatic test generation. In this paper, we applied our approach to a France Telecom[1] *Travel* service in order to demonstrate its feasibility.**

***Keywords*-Formal Methods; Timed Extended Finite State Machines; Nomad Language; Test Generation;**

I. Introduction

Security and reliability are important issues in designing and building systems with time constraints because any security failure can be risky for users, their business and/or their environment. Currently, software engineers developing systems, with time constraints, are not only confronted to their functional requirements but they also have to manage other aspects concerning security issues. We mean by 'functional requirements' the services that a system offers to end users. Whereas, security rules denote the properties (restrictions) that a system has to fulfill to be always in a safe state, or also to guarantee good quality of services it provides. For instance, a file system may have to specify the prohibition for a user to access to a specific document if he/she is not authenticated or if his/her session of 10 minutes has been expired. More generally, a security rule expresses the obligation, permission or interdiction to perform an action under given conditions called context.

Complex systems are often incrementally designed. They are evolutionary where new requirements may appear during its life cycle and then have to be integrated to its initial specification. In this paper, we deal with a particular kind of requirements denoting security properties. Basically, we provide a formal approach to integrate elaborated security rules involving time constraints into a system specification based on communicating extended timed automata called TEFSM specification [1].

The analysis of security issues led researchers and security experts to define of a big number of security languages and models that provide a formal representation of system's security policies. With the great majority of these models, security rules are specified using modalities like permission, prohibition and obligation that express some constraints on the system behavior. Among these models, we can mention for instance the Policy Description Language (PDL) [2], Ponder [3], RBAC (stands for Role Based Access Control) [4] and Nomad [5] (stands for Non atomic actions and deadlines). This latter is the one we have chosen since it allows to easily expressing security policies with time constraints.

In the literature, previous work dealt with secure systems specifications based on integration methodology. The authors of [6] and [7] for instance proposed formal procedures that let it possible to augment a functional description of a system with security rules expressed with OrBAC language [8] (stands for Organizationnal Based Access Control language). In both work, they described security rules that specify the obligation, permission or interdiction for a user to perform some actions under a given context. However, this context does not involve time aspects. In fact, they only specified rules without time considerations. Another work presented in [9] proposed to translate security rules (always without time constraints) into observers that can communicate with the functional specification of a system specified in EFSM formalism [10] (stands for Extended Finite State Machine) to

The research leading to these results has received funding from the European Community's Seventh Framework Program (FP 7/2007-203) under the grant agreement number 215995 (http://www.shields-project.eu/.)

[1]France Telecom is the main telecommunication company in France

regulate its behavior. The proposal presented in this article goes farther by considering time constraints that are very relevant in the modern applications that are more and more time-dependent. The main contributions of this paper are:

- The classification of the timed security rules into three distinguished classes. The first two classes denote basic rules including both atomic and non-atomic actions with simple contexts. A simple context only includes a single timed operator and neither logical nor structural connectors. The third class is general and deals with elaborated security rules that include more complex contexts involving several logical or/and structural connectors.
- The algorithms to integrate security rules within a TEFSM specification. These algorithms consist in adding or modifying guards, transitions and even states to make the execution instance of a given action possible only under a specific clock valuation. The produced TEFSM is called a secured TEFSM since it takes the timed security rules into account.
- The correctness proof of one integration algorithm which demonstrates the accuracy of our approach.
- An application of the proposed approach on an industrial case study provided by France Telecom company.

This paper is organized as follows. Section II provides the basic concepts used for the modeling of system behavior from both functional and security point of view. In section III, we present an overview of the approach we have developed to augment a TEFSM specification with timed security rules. Then, we describe the integration algorithm for basic security rules in section IV. The proof of the proposed algorithms is presented in section V. To demonstrate the feasibility of the developed approach, we present in section VI an industrial case study that deals with a business travel reservation web application provided by France Telecom. Finally, section VII concludes the paper.

II. Preliminaries

A. Modeling communicating systems using TEFSM model

The objective of modeling a system is to provide an operational specification of a system. This operational specification may include time constraints. In particular, it provides a better common understanding of the system. In addition, this operational model can also be used as input to existing validation tools, such as interactive or random simulators, model-checkers or (conformance) test generation engines. In our case, we rely in this paper on TEFSM model supported by IF language [1] because it includes the main concepts to design real-time systems. Moreover, several tools allowing its simulation and the generation of test sequences exist and are open source. A TEFSM modeling of a system consists of a set of IF processes (i.e entities), each one denotes a TEFSM that can communicate with other processes via FIFO channels.

Definition 1: A TEFSM M is a 7-tuple $M = < S, s_0, I, O, \vec{x}, \vec{c}, Tr >$ where S is a finite set of states, s_0 is the initial state, I is a finite set of input symbols (messages possibly with parameters), O is a finite set of output symbols (messages possibly with parameters), $\vec{x}$ is a vector denoting a finite set of variables, $\vec{c}$ is a vector denoting a finite set of clocks and Tr is a finite set of transitions. Each transition tr is a 4-tuple $tr =< s_i, s_f, G, Act >$ where s_i and s_f are respectively the initial and final state of the transition, G is the guard which denotes a predicate (boolean expression) on variables $\vec{x}$ and clocks $\vec{c}$ and Act is an ordered set (sequence) of atomic actions including inputs, outputs, variable assignments, clock setting, process creation and destruction.

The execution of any transition is spontaneous i.e. the action(s) associated with this transition occur simultaneously and take no time to complete. The time progress takes place in some states before executing the selected transitions. More details about time progress can be found in [11].

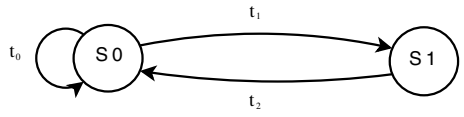


Figure 1. Example of a simple TEFSM with two states.

$t_0 =< S_0, S_0, P, (input\ a;\ T'\ ;output\ x) >$
$t_1 =< S_0, S_1, \bar{P}, (input\ a;\ T\ ;set\ Ck := 0\ ;output\ x) >$
$t_2 =< S_1, S_0, when\ Ck > 6, (input\ b;\ T"\ ;output\ y) >$

We illustrate the notion of TEFSM through a simple example depicted in Figure 1. This TEFSM is composed of two states S_0, S_1 and three transitions that are labeled with two inputs a and b, two outputs x and y, one guard (or predicate) P on variables, one clock Ck and three tasks T, T' and T''. The TEFSM operates as follows: starting from state S_0, when the input a occurs, the predicate P is checked. If the condition holds, the machine performs the task T, starts clock Ck, triggers the output x and moves to state S_1. Otherwise, the same output x is triggered but it is action T' that is performed and the state loops on itself. Once the machine is in state S_1, it can come back to state S_0 when the clock exceeds the value 6 and receives the input b. If so, task T'' is performed and output y is triggered.

Notations: In the remainder of the paper, we need the following notations. For each action a that belongs to the sequence actions Act:

- $before(a)$ denotes the actions of Act that are executed before action a. $before(a)$ is empty if action a is the first action of Act.
- $after(a)$ denotes the actions of Act that are executed after action a. $after(a)$ is empty if action a is the last action of Act.

- The sequence of actions Act can be then denoted by: $(before(a); a; after(a))$.

For instance, the action of transition t_1, of Figure 1, can be denoted by:

$$Act = (before(T); T; after(T))$$

where $(before(T) = input\ a)$ and $(after(T) = (set\ Ck := 0; output\ x))$.

B. Security rules specification using Nomad language

We use the Nomad formal language to specify without any ambiguity the set of security properties that the system has to respect. The choice of this language is mainly motivated by the Nomad features that provide a way to describe permissions, prohibitions and obligations related to non-atomic actions within elaborated contexts that takes into account time constraints. By combining deontic and temporal logics, Nomad allows to describe conditional privileges and obligations with deadlines, thanks to the time concept it supports. Finally, it can also formally analyze how privileges on non atomic actions can be decomposed into more basic privileges on elementary actions.

1) Nomad syntax and semantics: To meet the requirements of the functional model of the system, we define an atomic action in the Nomad language with the same concepts as for TEFSM actions.

Definition 2: We define an atomic action as one of the following actions: a variable assignment, a clock setting, an input action, an output action, a process creation or destruction.

Definition 3: (Formulae) If A is an action then $start(A)$ (starting A), and $done(A)$ (finishing A) are formulae.

- *If α and β are formula then $\neg\alpha$, $(\alpha \wedge \beta)$ and $(\alpha \vee \beta)$ are formula.*
- *If α is a formulae then $O^d\alpha$ (α was true d units of time ago if $d \leq 0$, α will be true after d units of time if $d \geq 0$) is a formulae too.*
- *If α is a formulae then $O^{<d}\alpha$ (within d units of time ago, α was possibly true if $d \leq 0$, α will be possibly true within a delay of d units of time if $d \geq 0$) is a formulae.*
- *If α and γ are formula then $(\alpha|\gamma)$ is a formulae whose semantics is: in the context γ, the formulae α is true.*

In the remainder of the paper, we respectively refer to operators "O" and "|" by timed and contextual operators. We also use the notation $O^{[\leq]d}$ to cover both cases O^d and $O^{<d}$. Notice also that using Nomad formalism, we deal with a discrete time. The choice of the unit of time can be very important and depends on the studied system. In our work, we use real time units like seconds, milliseconds or microseconds depending on the desired precision.

Definition 4: (A security rule) If α and β are formulae, $\mathcal{R}$ $(\alpha \mid \beta)$ is a security rule where R denotes one of the following deontic operators: $\{\mathcal{P}, \mathcal{F}, \mathcal{O}\}$. The security rule $\mathcal{P}$ $(\alpha \mid \beta)$ (resp. $\mathcal{F}$ $(\alpha \mid \beta)$, $\mathcal{O}$ $(\alpha \mid \beta)$) means that it is permitted (resp. prohibited, mandatory) to have α true when context β holds.

2) Examples of security rules specification: We present in this section some examples of security rules specifications expressed in Nomad:

Example 1:

$$\begin{array}{c}\mathcal{P}(start\ (input\ ReqWrite(user, file.doc))| \\ O^{\leq -5s}\ (done\ (output\ AuthOK(user))\wedge \\ \neg\ done\ (output\ DisconnectOK(user)))\end{array}$$

This rule expresses a permission granted to any user to request to write in 'file.doc', if earlier within 5 seconds, he/she was authenticated and his/her authentication is still running.

Example 2:

$$\begin{array}{c}\mathcal{O}(start\ (output\ DisconnectOK(user))| \\ O^{\leq -30min}(\neg\ done\ (input\ Message(user)))\wedge \\ O^{-30min} done(output\ AuthOK(user)))\end{array}$$

According to this obligation rule, the system must disconnect a running connection of any user if this latter remains inactive for 30 minutes.

Example 3:

$$\begin{array}{c}\mathcal{F}(start\ ((output\ AuthOK(user)))| \\ O^{\leq -0.01s}\ done\ (output\ AuthOK(user))\ \wedge \\ (\neg\ done(output\ DisconnectOK(user))))\end{array}$$

This prohibition rule means that it is forbidden that the system manages more than two authentication requests in the same millisecond.

III. Security integration methodology

The integration of security rules into a TEFSM model describing the behavioral aspects of a system leads to a TEFSM specification that takes the security policy into account: we call it 'secure functional specification'. The integration process is twofold. At first, the algorithm seeks for the rules to be applied on each transition of the TEFSM specification. Then, it adds some states, transitions or updates the guard of the related transition. These modifications depend on the nature of the rule (prohibition, permission or obligation) and its syntax format. To integrate security rules into a TEFSM specification, we have to make the following assumption: the security rules to integrate are consistent. We assume that it does not contain any incoherent or redundant rules. The consistency of the security policy is out of the scope of this paper and we assume that it has already been checked following different techniques (see for instance [12]). Here is an example of an inconsistent security policy composed of two rules $\mathcal{O}(start(A)|O^{-d}done(B))$ and $\mathcal{F}(start(A)|O^{-d}done(B))$. We cannot oblige the system to

perform action A in a context $(C = O^{-d}done(B))$ if this action is forbidden in the same context.

A. Security rules classification

According to Nomad syntax, there are several possible forms for security rules. It would obviously be tedious to deal separately with each of these forms. Consequently, we classify the Nomad security rules into two main classes described hereafter:

(1) Basic security rules: we consider in this class security rules of the form $\mathcal{R}(start(A)|O^{[<]d}done(B))$ where A and B are actions and where $\mathcal{R} \in \{\mathcal{F}, \mathcal{O}, \mathcal{P}\}$. To make easier the integration of such rules, we also distinguish two subclasses:
   - Basic security rules with atomic actions: actions A and B are atomic.
   - Basic security rules with decomposable actions. A or B or both are non-atomic actions.

(2) General security rules: a general security rule denotes any rule that does not fit into the first class. This means that the rule may contain several contextual or/and timed operators or/and logical connectors. In [13], we have demonstrated that such a rule can be decomposed and rewritten into one or several basic rules. In this way, integration algorithms developed for the first class can be reused and applied to integrate general security rules.

For sake of space, only algorithms to integrate prohibition and obligation timed security rules with basic actions are presented. A more complete description of the algorithms to deal with rules that include non-atomic actions several contextual or/and timed operators or/and logical connectors can be found in [14].

IV. Integration of basic security rules involving atomic actions

This section describes the integration of basic security rules of the form $\mathcal{R}(start(A|O^{[\leq]d}done(B))$ where $\mathcal{R} \in \{\mathcal{F}, \mathcal{O}, \mathcal{P}\}$, A and B denote atomic actions . Since we deal with a timed context, we need to define a global clock gck to manage the temporal aspect of the rules.

A. Prohibitions integration: $\mathcal{F}(start(A|O^{[\leq]d}done(B)))$

The prohibited action A is usually related to an already existing action in the initial system. A simple way to enforce the prohibition rule would be to disable all transitions containing action B (resp. A). In this way, there is no transition perfoming action B (resp. action A) anymore and the rule is obviously taken into account and respected. However, the main drawback of this solution is that it eliminates some behaviors that do not violate the security rule. In fact, the semantics of the rule does not forbid the execution of action A if action B is executed, for instance, $(d+1)$ units of time ago. Consequently, the key idea of integrating such prohibition rule in a TEFSM model is to check the rule context before performing the prohibited action. If this context is verified, the prohibited action A must be skipped. Otherwise, if the context is not valid, the action is performed without any rule violation.

Algorithm 1 Prohibitions Integration

Require: The TEFSM model $M =< S, s_0, I, O, \vec{x}, \vec{c}, Tr >$ and the prohibition security rule $\mathcal{F}$ *(start(A)* $\mid O^{[<]-d}$ *done(B))*

1: **for each** (transition tr such that $(tr \in Tr \wedge tr =< S_i, S_j, G, Act >)$) **do**
2: **if** $(B \in Act)$ **then**
3: $tr := < S_i, S_j, G, \{before(B), B, Update(Prohib(A)), after(B)\} >$;
4: **if** $(A \in after(B))$ **then**
5: /*$tr = < S_i, S_j, G, \{before(B), B, after(B) \cap before(A), A, after(A)\} >$*/
6: /*Create a new state S' and a new transition tr'*/
7: $tr := < S_i, S', G, \{before(B), B, after(B) \cap before(A)\} >$;
8: $tr' := < S'_k, S_j, \{when\ (\neg Prohib(A))\}, \{A, after(A)\} >$;
9: **end if**
10: **else**
11: **if** $(A \in Act)$ **then**
12: $tr := < S_i, S_j, G \wedge \{when\ (\neg Prohib(A))\}, Act >$;
13: **end if**
14: **end if**
15: **end for**

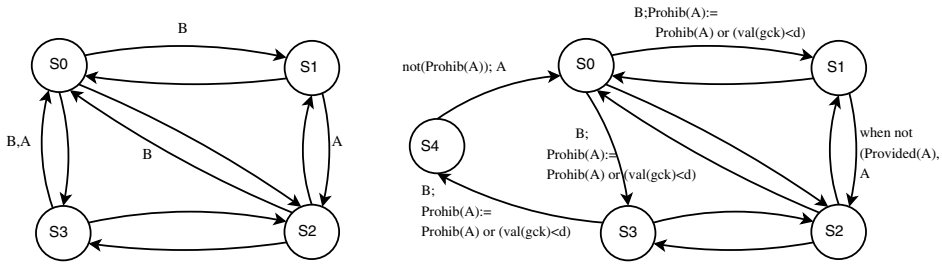
To achieve this goal, we have to define a table $Prohib$ to store all the instants where it is prohibited to execute a given action. These instants are denoted by a predicate on clock gck which is a global clock for the system lauched in its initial state. For instance, a predicate $((gck \leq 10) \vee (gck = 15))$ means that the execution of action A is forbidden till the tenth period of time and also at the fifteenth. We also defines the function $val(gck)$ that provides the value of clock gck at a specific moment. Table $Prohib$ is updated as follows:

- After each occurrence of B in the TEFSM transitions, value of $Prohib(A)$ is updated by adding a predicate on the instant(s) when the execution of action A is prohibited. The new value of $Prohib(A)$ is defined by:

$$Prohib(A) = \begin{cases} Prohib(A) \vee (gck < val(gck) + d), \\ \qquad for \mathcal{F}(start(A)|O^{<-d}done(B)) \\ Prohib(A) \vee (gck = val(gck) + d), \\ \qquad for \mathcal{F}(start(A)|O^{-d}done(B)) \end{cases}$$

- Before performing the prohibited action A, we check whether the value $val(gck)$ satisfies $(Prohib(A))$ to deduce if A can be performed or not.

The prohibition integration methodology is described in pseudo-code in Algorithm 1. To illustrate this algorithm, we present an example of a prohibition rule integration in Figure 2. In the left, the initial functional system contains several occurrences of the atomic actions A and B. We want to integrate the rule $\mathcal{F}$ *(start(A)* $\mid O^{<-d}$ *done(B))* that stipulates that it is forbidden to perform action A if within d units



(a) Initial System Specification (b) Secure System Specification

Figure 2. Prohibition Rule Integration : $\mathcal{F}$ (start(A) | $O^{[<]-d}$ done(B))

of time ago, B was performed. Applying Algorithm 1, we obtain the secure system depicted in Figure 2.b.

B. Obligations integration

Since several obligation rules related to action A may be defined, we have to take the possible dependencies that may exist between them. In fact let us consider for instances rules $\mathcal{O}$ *(start(A)* | O^{-5} *done(B))* and $\mathcal{O}$ *(start(A)* | $O^{<-10}$ *done(C))* where action C is executed 3 units of times (less than 5 units of times) after the execution of B. The execution of action A five minutes later the execution of B permits to satisfy both rules at the same time. The idea behind this simple example is to check if it necessary to execute the mandatory rule for each execution of the context action. To integrate an obligation security rule in the TEFSM based system specification, we rely on a new process RHP that ensures the execution of the mandatory action. If the related mandatory action is not executed by the initial specification, the process has the task to execute it itself. We assume that the initial system S is not secure with respect to each obligation rule of the form $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))*, that is, it does not perform the action A, d units of time after executing B. This task is then performed by the RHP process. The integration methodology follows these steps for a rule that is in form of $\mathcal{O}$ *(start(A)* | $O^{[\leq]-d}$ *done(B))* where $d > 0$:

- A boolean variable $wait_A$ is defined. It allows to know whether we are waiting for the execution of an instance of action A or not. This variable is set to *true* at the execution of each action B for which an obligation rule $\mathcal{O}$ *(start(A)* | $O^{[\leq]-d}$ *done(B))*, and set to *false* when action A is executed.
- The definition of a new process that can be created n times by the initial functional specification. n is the maximum number of occurrences of action B in the initial TEFSM specification. This new process has two parameters. The first parameter, equal to $(val(gck)+d)$, states the instant when (resp. before which) action A should be executed if we deal with the O^{-d} timed operator (resp. $O^{<-d}$). The second parameter $exactTime$ is a boolean intended to know whether action A must be executed at $(gck + d)$ (for a rule of the form $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))*) units of times or before this moment (for a rule of the form $\mathcal{O}$ *(start(A)* | $O^{<-d}$ *done(B))*). The process has to wait until the deadline of execution of action A is reached. This deadline and the actions to perform $(deadline, Action)$ depend on the type of the rule and whether we are waiting for an execution of action A:

$(deadline, Action) =$
  for $\mathcal{O}(start(A)|O^{-d}done(B))$:
    $(gck = gck + d, A; (wait_A := false); stop)$
  for $\mathcal{O}(start(A)|O^{<-d}done(B))$:
    $((gck = gck + d) \vee \neg wait_A, if wait_A then A; stop)$

The complete algorithm that permits to integrate an obligation rule of the form $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))* [2] is as follows:

In Figure 3, we present the integration of an obligation rule within the initial system depicted in Figure 2.a. In this functional system, we can find several occurrences of the atomic action B.

V. Correctness proof of the integration approach

To ensure the correctness of the approach we have proposed to integrate security rules, we have to prove all the algorithm we have developed. For sake of space, we only present here the correctness of algorithm 1 for rules of the form $\mathcal{F}$ *(start(A)* | $O^{<-d}$ *done(B))*. Other correctness proofs are similar and can be found in [13].

By proving the correctness of algorithm 1, we precisely demonstrate that the integration of prohibition rule of the form : $\mathcal{F}$ *(start(A)* | $O^{[<]-d}$ *done(B)) where* $d > 0$ produces to a secure TEFSM specification. To achieve this goal, we define for each occurrence of action B:

- t_B as the instant of the execution action B.
- k as the time elapsed after the execution of action B.
- gck is a global clock of the system that gives the current time at each moment.

We have to prove that we can not perform the action A within d units of time after the execution of action B.

[2]For an obligation rule of the form $\mathcal{O}$ *(start(A)* | $O^{[<-d}$ *done(B))*, we have just to replace the second parameter of process RHP with *false*.

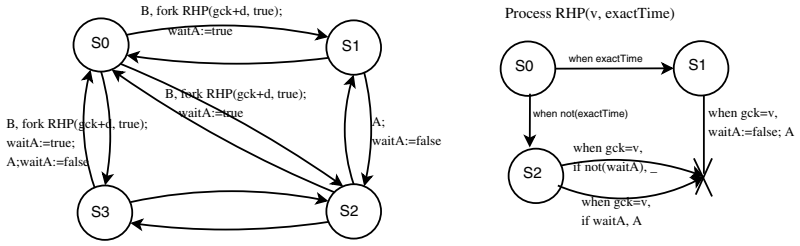


Figure 3. Obligation Rule Integration : $\mathcal{O}$ (start(A) | O^{-d} done(B)).

Algorithm 2 Obligations Integration

Require: The TEFSM model $M =< S, s_0, I, O, \vec{x}, \vec{c}, Tr >$ and the obligation security rule $\mathcal{O}$ *(start(A)* | $O^{<-d}$ *done(B))*

1: In the initial state of M, $wait_A := 0$
2: **for each** (transition tr such that $(tr \in Tr \wedge tr =< S_i, S_j, G, Act >)$) **do**
3: **if** $(B \in Act)$ **then**
4: $tr := < S_i, S_j, G, (before(B); B; wait_A := true; fork\ RHP(gck + d, true); after(B)) >$
5: **end if**
6: **if** $(A \in Act)$ **then**
7: $tr := < S_i, S_j, G, (before(A); A; wait_A := false; after(A)) >$
8: **end if**
9: **end for**
10: **for** RHP process (v, exactTime) **do**
11: Define two states S_1 and S_2
12: Define five transitions tr_1, tr_2, tr_3, tr_4 and tr_5
13: $tr_1 := < S_0, S_1, \{when\ exactTime\}, \_ >$
14: $tr_2 := < S_0, S_2, \{when\ not\ exactTime\}, \_) >$
/*We should execute action A even if an other instance of this action has already been executed */
15: $tr_3 := < S_1, \_, \{when\ gck = v\}, (wait_A := flase; A; stop) >$
/*We should execute action A only if no instance has been executed before $(v = gck + d)$*/
16: $tr_4 := < S_2, \_, \{when\ gck = v\}, (if(wait_A)A); stop)) >$
17: $tr_5 := < S_2, \_, \{when\ gck = v\}, (if(not(wait_A))); stop)) >$
18: **end for**

Mathematically, we have to establish that for each positive integer k the following predicate holds:

$$((k < d) \wedge (val(gck) = t_B + k)) \Rightarrow \neg start(A) \qquad (1)$$

To prove that action A cannot be executed at the moment $(val(gck) = t_B + k)$, it is sufficient to prove that at this moment all the transitions Tr of the secure system labeled by action A are impassable. That is all the guards of the transitions Tr are false. In the secure system, the algorithms 1 adds the guard $(when\ (\neg Prohib(A)))$ before each execution of A. Thus, we have to prove, for each positive integer k, that:

$$((k < d) \wedge (val(gck) = t_B + k)) \Rightarrow Prohib(A) \qquad (2)$$

Predicate $Prohib(A)$ is a disjunction of predicates to which predicate $(gck < val(gck) + d)$ has been added when action B has been executed at instance t_B. The added predicate is equivalent to $(gck < t_B + d)$ (See algorithm 1). Consequently, it is sufficient to establish that:

$$((k < d) \wedge (val(gck) = t_B + k)) \Rightarrow val(gck) < t_B + d \qquad (2)$$

which is obviously true.

VI. Case study: Travel Web application

A. Travel application functional description

To prove the effectiveness of our framework we carried out a case-study using a Travel application which is an internal service used by France Telecom company to manage 'missions' (business travels) carried out by its employees. In our case study we only consider, at first, a simple Travel application where a potential traveler can connect to the system (using a dedicated URL) to request a travel ticket and a hotel reservation during a specific period according to some business purposes (called mission). This request can be accepted or rejected by his/her hierarchical superior (called validator). In the case it is accepted, the travel ticket and hotel room are booked by contacting a travel agency. The specification of this Travel Web application is performed using the IF language that relies on TEFSM model.

B. Security integration for travel Web system

Further, we defined some specific security rules to boost the system security. These security rules are inspired from France Telecom's security test campaign in the context of POLITESS project[3]. The security rules are formally specified using the Nomad model.

1) Travel security specification using Nomad language: France Telecom proposed a preliminary version of the case study Travel in which some informal security requirements are provided. Based on these requirements, we formally specified a set of 34 security rules using the Nomad language. For matter of space, we only present the following three:

- Rule 1:

$$\mathcal{F}\ (start\ (output\ req\_create\_mission(t))| \\ O^{\leq -2min}\ done\ (output\ req\_create\_mission(t)))$$

[3]http://www.rnrt-politess.info/

This first prohibition rule expresses that two missions requests of the same traveler must be separated by at least 2 minutes. This request can be performed in the *basic_traveler* process.

- Rule 2:

$$\mathcal{P}\ (start\ (output\ req\_proposition\_list(t,m))| \\ O^{\leq -10min}\ done\ (output\ req\_proposition\_list(t,m)))$$

This permission rule expresses that a traveler can request for another list of travel propositions within a delay of 10 minutes if he/she already asked for a first list of travel propositions. This request can be performed in the *traveler_mission* process.

- Rule 3:

$$\mathcal{O}\ (start\ (output\ req\_validation()))| \\ O^{-10080min}\ done\ (output\ req\_validation()) \wedge \\ O^{\leq -10080min}\ ((\neg\ done\ (input\ recv\_validate\_notification())) \\ \wedge\ (\neg\ done\ (input\ recv\_unvalidate\_notification()))))$$

This obligation rule expresses that if a traveler requested for the validation of his/her mission and if he/she did not received an answer, the system must send, as a reminder, another request to the potential mission validator. This reminder is sent within a delay of (10080 min = 7 days). The requests and answers are made in the *travel_mission* process.

2) Rules integration results: A securuity rules integration module based on the integration methodology described in section IV has been implemented using C language. The table I shows some metrics about the modifications after the integration of some specific rules: the modified and added transitions (M&A Transitions), the added variables and clocks (Added Var & Ck), the added processes (Added Proc).

Table I
IF TRAVEL SYSTEM MODIFICATIONS ACCORDING TO EACH RULE

Rule	M&A Transitions	Added Var & Ck	Added Proc
1	1+1	1	0
2	2+1	1	0
3	4+3	1	1

C. Test Generation

To automatically generate test cases from the secure specification of Travel, we use the TestGen-IF [15] test generation tool. This tools implements a timed test generation algorithm based on a Hit-or-Jump exploration strategy [16]. This algorithm efficiently constructs test sequences with high fault coverage, avoiding the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random searches. It allows to produce a set of test scenarios according to a set of test purposes. The automatic test generation only targets security issues and, as a result, it is less time consuming.

To reach this aim we used, for Travel test generation, two users (one being the validator) and two missions. We also defined adequate interval values for data variables in order to reduce the accessibility graph size and to avoid state explosion problems.

A set of timed test cases are generated based on the IF specification of Travel Web application and the timed test purposes for each rule, using TestGen-IF. These test cases are then filtered according to the observable actions (input, output and delays) relating to the system under test. Some metrics about this test generation relating to three rules are presented in Table II.

D. Test Cases Instantiation and Execution

In order to execute the generated test cases to a real Web application, they need to be transformed into an executable script capable of communicating via http (or https) with the implementation under test. In this work, we automatically translated the abstract test case into tcl script that is used by the tclwebtest tool [17] to execute the designed tests.

The test cases execution was performed on a prototype implementation of the Travel Web application (developed on OpenACS platform) to verify that the specified security requirements are respected. It is important to highlight that some time settings in this prototype have been changed so that the application of the tests where faster than in the real system. For example, we changed 10080 minutes (7 days) in the third rule to 3 minutes to avoid waiting so long. Therefore in this case study we verify the behavior of the system concerning this rule using a delay of 3 minutes rather than using 7 days.

The execution of the test cases is performed using a dedicated testing tool proposed by the OpenACS community [18]. This tool is called the ACS-Automated-Testing tool that allows executing the instantiated test cases, interacting with the Web-based application under test and, also, displaying the verdict of each test case. The ACS-Automated-Testing tool is, in itself, a Web application but we will refer to it just as *the tester* to avoid confusions between this system and the Web application to be tested.

As a result of the execution of the designed test cases on the prototype, we obtained positive verdicts for thirty test objectives, while, four test objectives were violated (fail verdict). For example, a problem has been detected according to the system respect to the first rule that expresses a prohibition. If a potential traveler requests for a first mission and then waits for 2 minutes, he/she is not allowed by the system to request for another mission. We analyzed the implementation of the Web-based system and noticed that a mistake was encrusted in the code. Instead of 2 minutes, the Travel system waited much longer before allowing a second mission request.

The Travel application was analyzed to detect the four error sources. Once the mistakes corrected, all the test cases were applied again on the Web application. This time, all

Table II
SOME TEST GENERATION METRICS

Rule	Strategy	Maxdepth	Jumps	Test Case Length	Visited States	Duration
1	BFS	10	0	9	291	0.2s
2	BFS	10	1	16	7844	10s
3	BFS	10	2	23	26552	1m25s

the verdicts were positive which demonstrates the efficiency of our methodology.

VII. CONCLUSION

In this paper, we have presented a formal approach to integrate timed security rules, expressed according to Nomad language, into a TEFSM specification of a system. Roughly speaking, a security rule denotes the prohibition, the obligation or the permission for the system to perform an action in a specific timed context. To meet this objective, we have described a set of algorithms that allows to add them to a TEFSM specification describing the behavior aspect of a system. A proof that demonstrates the correctness of the prohibition integration algorithms is given. These algorithms are implemented in a tool and the methodology has been applied on several real-size applications that gave very promising results. Finally, the complexity of the algorithms presented in this paper are linear (in $O(n)$) since it is directly proportional to the number of rules in the policy. In practical experience, the modification of an initial system is not sizeable since the number of rules is not in general huge. In most transitions, only some changes in predicates are applied.

REFERENCES

[1] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," in *SFM*, 2004, pp. 237–267.

[2] J. Lobo, R. Bhatia, and S. A. Naqvi, "A Policy Description Language," in *AAAI/IAAI*, 1999, pp. 291–298.

[3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "Ponder: An object-oriented language for specifying security and management policies," in *10th Workshop for PhD Students in Object-Oriented Systems (PhDOOS'2000), 12-13 June 2000, Sophia Antipolis, France*, 2000, extended Abstract. [Online]. Available: citeseer.ist.psu.edu/466775.html

[4] J. F. Barkley, K. Beznosov, and J. Uppal, "Supporting Relationships in Access Control Using Role Based Access Control," in *ACM Workshop on Role-Based Access Control*, 1999, pp. 55–65.

[5] F. Cuppens, N. Cuppens-Boulahia, and T. Sans, "Nomad: A Security Model with Non Atomic Actions and Deadlines," in *CSFW*, 2005, pp. 186–196.

[6] W. Mallouli, J.-M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens, "A Formal Approach for Testing Security Rules." in *SACMAT*, Nice, France, 2007.

[7] N. Benaïssa, D. Cansell, and D. Méry, "Integration of Security Policy into System Modeling," in *B*, 2007, pp. 232–247.

[8] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin, "Organization Based Access Control," in *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, Lake Come, Italy, June 4-6 2003.

[9] K. Li, L. Mounier, and R. Groz, "Test Generation from Security Policies Specified in Or-BAC," in *COMPSAC (2)*, 2007, pp. 255–260.

[10] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - A Survey," *IEEE Transactions on Computers*, vol. 84, pp. 1090–1123, August 1996.

[11] M. Bozga, S. Graf, L. Mounier, and I. Ober, "IF Validation Environment Tutorial," in *SPIN*, 2004, pp. 306–307.

[12] L. Cholvy and F. Cuppens, "Analyzing Consistency of Security Policies," in *IEEE Symposium on Security and Privacy*, 1997, pp. 103–112.

[13] W. Mallouli, A. Mammar, and A. R. Cavalli, "Integration of Timed Security Policies within a TEFSM Specification," Telecom SudParis, Technical Report TI-PU-08-868, 2008.

[14] W. Mallouli, "A Formal Approach for Testing Security Policies," Ph.D. dissertation, Telecom and Management SudParis Evry-France, December 2008.

[15] A. R. Cavalli, E. M. D. Oca, W. Mallouli, and M. Lallali, "Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints," in *The 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Vancouver, British Columbia, Canada, October 27-29, 2008.

[16] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zadi, "Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services," in *Formal Methods for Protocol Engineering And Distributed Systems*, Beijing, China, october 1999, pp. 41–56.

[17] TCLWEBTEST Tool, "http://tclwebtest.sourceforge.net/."

[18] OpenACS Community, "http://www.openacs.org/."