Modeling System Security Rules with Time Constraints Using Timed Extended Finite State Machines*

Wissam Mallouli, Amel Mammar and Ana Rosa Cavalli
Institut Télécom / Télécom SudParis, CNRS/SAMOVAR, France
{wissam.mallouli, amel.mammar, ana.cavalli}@it-sudparis.eu

Abstract

Security and reliability are of paramount importance in designing and building real-time systems because any security failure can put the public and the environment at risk. In this paper, we propose a framework to take timed security requirements into account from the design stage of the system building. Our approach consists of two main steps. 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 properties specified in Nomad language. Nomad is a formal language well adapted to express timed security properties with timed constraints. We also briefly present a France Telecom[1] Travel system as a case study to demonstrate the reliability of our framework.

Keywords: Timed EFSM Model, Security Rules, Nomad Language, Security Integration.

1 Introduction

The last decade has witnessed a substantial proliferation of real-time systems in numerous domains. Such systems have a time dependent behavior as well as an increasing demand for security, which mainly due to their complexity and distribution aspects. Consequently, software engineers developing these systems are not only confronted to functional requirements but they have also to manage other kinds of requirement concerning security issues. Roughly speaking, by "functional requirements" we mean the services that a system has to offer to end users. Whereas, security rules denote the properties that a system has to fill to be always in a safe state or also to guarantee a good quality of the services it provides.

*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/), and POLITESS French project (http://www.rnrt-politess.info/).

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

Industrial systems are often designed in two steps. The first step consists in specifying the real time system from the functional point of view. Many models based on time automata theory [4, 19] are proposed in the literature to perform this formal specification including functional timed constraints. However, this specification has to be completed later by integrating the system security aspects. Unfortunately, software engineers developing industrial systems are not necessarily experts in security. As a consequence, managing security requirements is a main issue which is far from being easy for them.

To tackle this problem, we have introduced in an earlier publication [17] a formal process that permits to augment a functional description of a system with security rules expressed with OrBAC language [2] (stands for **Or**ganisationnal **B**ased **A**ccess **C**ontrol). We described security rules that specify the obligation, permission or interdiction for a user to perform some actions under given conditions called *context*. This context does not involve time aspects. In fact, we only specified rules without time considerations. Another work presented in [15] proposes to translate security rules (always without time constraints) into observers that can communicate with the functional specification of a system specified in EFSM formalism [14] (stands for **E**xtended **F**inite **S**tate **M**achine) to regulate its behavior. To be able to include timed security rules, for example the obligation for a client to send his/her payment within 8 days at the latest after the reception of his/her bill, we propose in this paper to rely on Nomad language [13] (stands for **N**on-atomic **a**ctions and **d**eadlines) that supports the time concept.

The main contribution of this paper is to provide software engineering with a formal process to integrate more elaborated security rules involving timed contexts into a TEFSM specification [19]. Such an integration consists in adding clocks to represent the time progress and also in adding guards (or predicates), transitions and/or 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.

From a theoretical point of view, the integration approach we have developed classifies security rules into three distinguished classes. The first two classes denote basic rules with atomic or non-atomic actions and whose contexts are simple. By a simple context, we mean a context that includes a single timed operator and no logical connectors. The third class is general and deals with elaborated security rules that include more complex contexts. We can demonstrate that these last rules can be decomposed into one or several basic rules on which the integration process defined for the two first classes can be applied [16]. For the sake of space, we only present the integration of basic security rules including atomic actions. The other cases are presented in details in a technical report [16]. It is worth noticing that the integration of security rules within a functional system specification is not an end in itself. The secured TEFSM can be used for several purposes such as code generation [18], specification correctness proof[3], model checking [11] or automatic test generation [1, 9], etc.

This paper is organized as follows. Section 2 provides the basic concepts used for the modeling of system behavior from functional and security point of view. In section 3, we expose the algorithms to integrate basic security rules within an existing TEFSM specification. The application of our methodology to an industrial case study is presented in section 4. Finally, section 5 concludes the paper and presents future work.

2 Preliminaries

2.1 Modeling Communicating Systems Using TEFSM Model

The objective of modeling a system is to provide an operational specification of a system from a functional point of view which can include time constraints. In particular, it helps to provide 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. To achieve this modeling goal, we rely in this paper on TEFSM model [19]. A TEFSM modeling of a system consists of a set of processes, each process 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. A 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 is composed of predicates on variables and clocks $\vec{x}$ and $\vec{c}$ (boolean expression) 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 in the sense that 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.

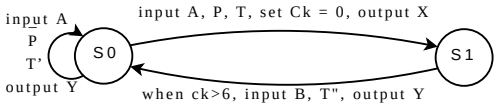


Figure 1. Example of a simple TEFSM with two states.

We illustrate the notion of TEFSM through a simple example described in Figure 1. This TESFM 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 the clock Ck, triggers the output X and moves to state S_1. Otherwise, the same output Y 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.

In the following sections, if $tr = < s_i, s_f, G, Act >$ and $a \in Act$, then we can denote Act by $\{before(a), a, after(a)\}$ to express that action a is performed within the transition tr and that there is possibly other actions before or after a ($before(a)$ and $after(a)$ may be empty). For instance, in the transition from state S_0 to S_1 of Figure 1, $Act = \{input\ A, T, set\ Ck := 0, output\ X\}$ can be denoted by $Act = \{before(T), T, after(T)\}$ where $before(T) = \{input\ A\}$ and $after(T) = \{set\ Ck := 0, output\ X\}$.

2.2 Security Rules Specification Using Nomad Language

We rely in this paper on Nomad formal language to specify without any ambiguity the set of security properties that the system has to respect. The choice of this language was

mainly motivated by the characteristics of Nomad that provides a way to describe permissions, prohibitions and obligations related to non-atomic actions within elaborated contexts and mainly 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.

2.2.1 Nomad Syntax and Semantics

To meet the requirements of the functional model of the system, we define an atomic action with the same concepts of TEFSM actions.

Definition 2 *(Atomic action) 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 a process destruction.*

Definition 3 *(Non-atomic action) If A and B are actions, then $(A; B)$, which means "A is followed immediately by B" is a non-atomic action.*

Definition 4 *(Formulae) If A is an action then $start(A)$ (starting A), and $done(A)$ (finishing A) are formulae.*

Here are some properties on actions and formulae:

- If α and β are formulae then $\neg\alpha$, $(\alpha \wedge \beta)$ and $(\alpha \vee \beta)$ are formulae.
- If α is a formula 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 formula too.
- If α is a formula 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 formula.
- If α and γ are formulae then $(\alpha|\gamma)$ is a formula whose semantics is: in the context γ, the formula α is true.

In the rest of the paper, we use the notation $O^{[<]d}$ 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 5 *(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 execute α when context β holds.*

More details about the syntax and semantics of Nomad are presented in [13].

2.2.2 Examples of Security Rules Specification

We present in this section some examples of security rules specifications expressed in Nomad:

Example 1:

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

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:

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

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:

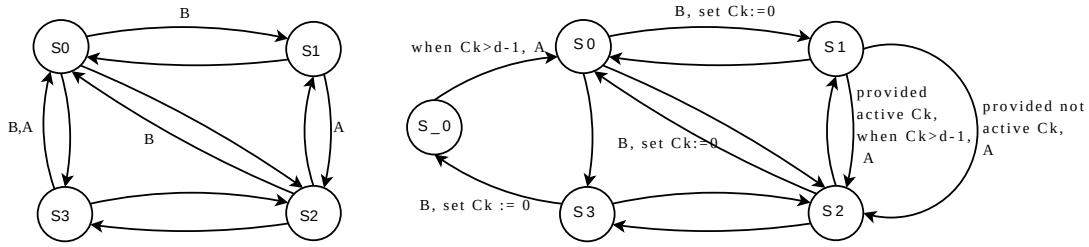
$$\mathcal{F}(start\ ((output\ AuthOK(user)))| \\ O^{\leq -0.01s}\ done\ (output\ AuthOK(user))\ \wedge \\ (\neg\ done(output\ DisconnectOK(user))))$$

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

3 Security Integration Methodology

In this section, we present our approach to integrate security rules into a TESFM specification describing the behavioral aspects of a system. We deal with rules of the form $\mathcal{R}\ (start(A|O^{[\leq]d}done(B))$ where actions A and B are atomic actions. To integrate security rules into a TEFSM specification, we have to make the following assumptions:

- The initial TEFSM specification representing the behavior of the system is correct. Indeed, it must be deadlock free and each state must be reachable from any other state.
- The initial TEFSM specification of the system does not take into account the security requirements we would like to integrate. It only specifies the system behavior from its functional point of view. Such a property can be verified using for instance model checking methods [11]. If the rule is already satisfied by the initial specification, there is no need to add it again.



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

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

- The security rules to integrate are consistent. We assume that it does not contain any incoherent or redundant rules. Checking the consistency of the security policy is out of the scope of this paper. We assume that this issue has been checked. There are several techniques to achieve this goal (see for instance [12]). Here is an example of 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 can not oblige the system to perform action A in a context ($C = O^{-d}$ done(B)) if this action is forbidden in the same context.

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

The prohibited action usually relates to an already existing action in the initial system. Considering the TEFSM specification, action B can appear on one or several transitions. The basic 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 can be performed without any rule violation. Since we deal with a timed context, we have to define a clock to manage the temporal aspect of the rule.

First Case: in the following, we present the different steps to integrate a prohibition rule in the form of $\mathcal{F}$ *(start(A) |* $O^{<-d}$ *done(B))* within a TEFSM model where ($d > 0$). This rule expresses that it is forbidden to perform action A if within ($d-1$) units of time ago, B was performed. Three steps are to be considered:

- The creation of a public clock Ck that can be modified by all the TEFSM model processes.
- Clock Ck is set to 0 after each occurrence of B in the TEFSM. Intuitively, Ck measures the time elapsed from the last execution of action B. Before the first execution of B, clock Ck is simply inactive.
- Before performing the prohibited action A, we verify if clock Ck is already activated. If so, we check its valuation to deduce if A can be performed or not. If the clock is not activated, that means that the system did not perform B yet. In that case, A is allowed.

These steps are provided 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) |* $O^{<-d}$ *done(B))* that stipulates that it is forbidden to perform action A if within d units of time ago, B was performed. Applying Algorithm 1, we obtain the secure system depicted in Figure 2.b.

Algorithm 1 Prohibitions Integration (1/2)

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

1: Define a new integer variable k:=0;
2: Define a new clock Ck within M;
3: **for each** (transition tr such that $(tr \in Tr \wedge tr =< S_i, S_j, G, Act >)$) **do**
4: **if** ($B \in Act$) **then**
5: $tr := < S_i, S_j, G, \{before(B), B, set\ Ck := 0, After(B)\} >$;
6: **if** (($A \in Act$) $\wedge$ $A \in After(B)$) **then**
7: /*$tr = < S_i, S_j, G, \{before(B), B, After(B) \cap Before(A), A, After(A)\} >$*/
8: Create a new state S'_k and a new transition Tr_k;
9: $tr := < S_i, S'_k, G, \{before(B), B, After(B) \cap Before(A)\} >$;
10: $tr_k := < S'_k, S_j, \{when\ (Ck > d-1)\}, \{A, After(A)\} >$;
11: k++;
12: **end if**
13: **else**
14: **if** ($A \in Act$) **then**
15: Create a new transition tr_k;
16: $tr := < S_i, S_j, \{G, provided\ not\ active\ Ck\}, \{before(A), A, After(A)\} >$;
17: $tr_k := < S_i, S_j, \{G, provided\ active\ Ck, when\ (Ck > d-1)\}, \{before(A), A, After\ (A)\} >$;
18: k++;
19: **end if**
20: **end if**
21: **end for**

Second Case: this part gives the steps to follow in order to integrate, within a TEFSM specification, a prohibition rule of the form $\mathcal{F}$ *(start(A)* | O^{-d} *done(B))* where $d > 0$. This rule expresses that it is forbidden to perform action A if B was performed d units of time ago. The first solution that comes to mind consists in defining -like in the first case- a new clock Ck which is set to 0 each time action B is executed. Then, the guard of each transition that executes action A is reinforced by the guard $\{ck \neq d\}$ to make the transition fireable only if the elapsed time from the execution of action B is different from d (It may be more or less). This solution is represented by a declination of Algorithm 1 by replacing $\{when\ Ck > d-1\}$ with $\{when\ Ck \neq d\}$.

Figure 3 illustrates the application of this algorithm on the example of the initial TESM presented in Figure 2.a. Basically, we have to check the valuation of the clock Ck to know whether it is permitted or not to execute A.

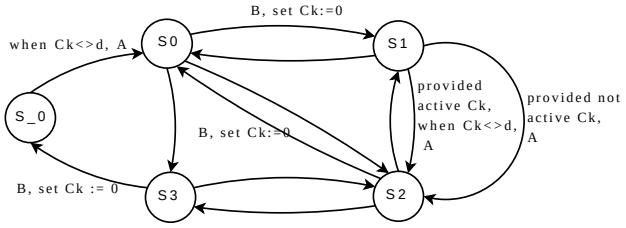


Figure 3. First Intuition for Prohibition Rule Integration

However, a deep analysis about the presented solution shows that this latter is only conceivable if the interval between two successive executions of action B is longer than d. Indeed, let us assume that TEFSM system in Figure 3 follows the sequences of transitions shown in Table 1 and that clock Ck progresses after its activation in each state S_i according to a given valuation.

	Transition	Arrival State S_i	Duration in S_i
Tr_1	$S0 \rightarrow S2$	$S2$	2
Tr_2	$S2 \rightarrow S0$	$S0$	3
Tr_3	$S0 \rightarrow S1$	$S1$	2
Tr_4	$S1 \rightarrow S2$	$S2$	Not relevant

Table 1. A Transitions Sequence Example with Time Progress

Let us suppose that d is equal to 5. gck denotes a master clock that measures the system global time. The progress of the secure system is described in Table 2.

We can notice that since Ck is not equal to 5, action A is 'wrongly' executed although the time elapsed from the first execution of action B is equal to 5. This is due to the reset action ($Ck := 0$) executed in the second occurrence of B. In other words, this re-set action erases the previous possible execution of B from the system memory.

Transition	States	gck	Ck	Note
Tr_1	$S0$	0	-1	Ck is not yet activated
	$S2$	0	-1	Transitions are instantaneous
Time progress (2 units of time)				
Tr_2	$S2$	2	-1	Ck is not yet activated
	$S0$	2	0	1st execution of B
Time progress (3 units of time)				
Tr_3	$S0$	5	3	Both clocks progress
	$S1$	5	0	2nd execution of B
Time progress (2 units of time)				
Tr_4	$S1$	7	2	Both clocks progress
	$S2$	7	2	Action A is performed since $Ck \neq 5$

Table 2. The Secure TEFSM System Progress

To cope with this limit, we suggest the following second solution. Basically, we define a clock gck that denotes a master clock that measures the time elapsed from the beginning and an integer variable c that indicates the next moment when the execution of A is forbidden. Thus for each execution of action B, the system creates a new process RHP (for Rule Handler Process) that waits during d units of time. Then, it updates the value of c to state the moment when the execution of A is forbidden, then it stops (it kills itself). The global clock gck is compared to the value of c before performing A. The algorithm 2 formally defines these steps. Applying this algorithm on the TEFSM of Figure 2.a gives the secured TEFSM depicted in Figure 4.

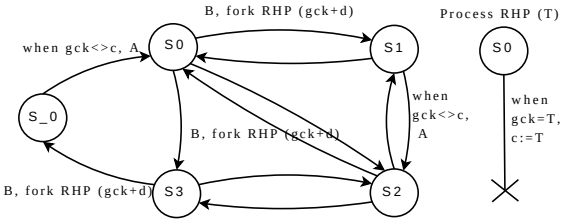


Figure 4. Prohibition Rule Integration : $\mathcal{F}$ (start(A) | O^{-d} done(B)).

3.2 Permissions Integration

Like prohibitions, permissions relate to actions which already exist in the initial functional system. Their integration within a TEFSM specification relies on almost the same algorithms for prohibition rules integration presented in the last section. Indeed, by definition, giving the permission to perform an action A in a given context C, means that when this context is not verified, A must be denied. Thus, a permission rule $\mathcal{P}$ *(*α | β*)* is equivalent to the prohibition rule $\mathcal{F}$ *(*α | $\neg\beta$*)*. The algorithms used for the integration of permissions rules are very similar to algorithms 1 and 2. More details are provided in [16].

Algorithm 2 Prohibition Integration (2/2)

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

1: Define a new integer variable k:=0;
2: In M, define a new public clock gck and a new public integer variable c;
3: In the initial State, set gck := 0; c := -1;
4: **for each** (transition tr such that $(tr \in Tr \wedge tr =< S_i, S_j, G, Act >)$) **do**
5: **if** $(B \in Act)$ **then**
6: $tr :=< S_i, S_j, G, \{before(B), B, fork\ RHP\ ((integer)gck + d), After(B)\} >$;
7: /*RHP is a new process that handles variable c. It accepts an integer parameter*/
8: **if** $((A \in Act) \wedge A \in After(B))$ **then**
9: /*$tr =< S_i, S_j, G, \{before(B), B, After(B) \cap Before(A), A, After(A)\} >$*/
10: Create a new state S'_k and a new transition tr_k;
11: $tr :=< S_i, S'_k, G, \{before(B), B, After(B) \cap Before(A)\} >$;
12: $tr_k :=< S'_k, S_j, \{when\ gck \neq c\}, \{A, After(A)\} >$;
13: k++;
14: **end if**
15: **else**
16: **if** $(A \in Act)$ **then**
17: $G := \{G, when\ gck \neq c\}$;
18: **end if**
19: **end if**
20: **end for**
21: **for** RHP (T) **do**
22: In the initial state S_0, define a transition $tr1$;
23: $tr1 :=< S_0, \_, when\ gck = T, \{c := T, stop\} >$;
24: **end for**

3.3 Obligations Integration

To integrate an obligation security rule, 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.

First Case: the integration methodology follows these steps for a rule that is in form of $\mathcal{O}$ *(start(A)* | O^{-d} *done(B))* where $d > 0$:

- 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 the rule context action B that can be executed during d units of time.
- The new process has to set a clock and wait until the deadline is reached. At this moment, it performs the mandatory action A.

We assume that the initial system S is not secure, 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.

In Figure 5, 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.

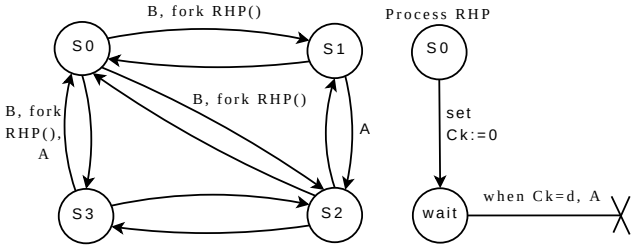


Figure 5. Obligation Rule Integration : $\mathcal{O}$ (start(A) | O^{-d} done(B)).

Second Case: to add an obligation rule of the form of $\mathcal{O}$ *(start(A)* | $O^{<-d}$ *done(B))*, we have to associate with each occurrence of action B an execution of action A. This latter action has to be performed within a delay of $(d-1)$ units of time. To perform such an integration, we have to follow the steps given hereafter:

- We define an integer variable $wait_A$ that counts the number of occurrences of actions B that are waiting for an execution of action A.
- We define a new process RHP where a clock Ck is activated to wait $(d-1)$ units of time (till action A has to be executed). When the deadline is reached, process RHP checks whether we are waiting for any execution of action A ($wait_A > 0$) and executes A if necessary.
- Variable $wait_A$ are updated as follows: $wait_A$ is incremented each time action B is executed. If the value $wait_A$ is strictly positive, it is decremented each time action A is executed either by the initial specification or by process RHP.

Intuitively, process RHP has to wait for a possible execution of action A during the allowed time $(0..(d-1))$. In case where the initial specification does not execute such an action, process RHP must execute it. Figure 6 shows the integration of obligation rule of the form $\mathcal{O}$ (start(A) | $O^{<-5}$ done(B)) within the initial system shown in Figure 2.a.

4 Case-study: Travel Application

To prove the effectiveness of our framework, we carried out a case-study on a travel application which is an internal service used by France Telecom company to manage 'missions' corresponding to traveling of its employees. In our case study, we only consider at first a simple travel application where a potential traveler can connect to system

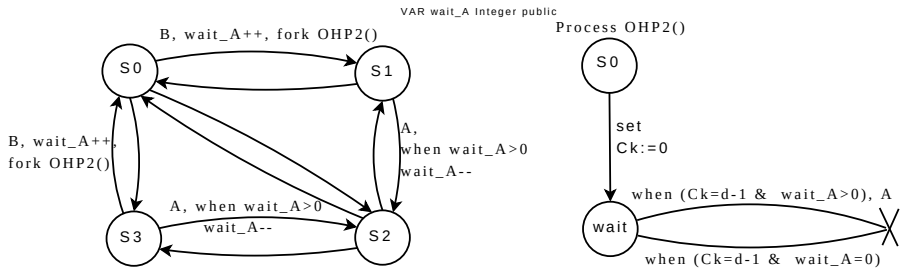


Figure 6. Obligation Rule Integration : $\mathcal{O}$ (start(A) | $O^{<-d}$ done(B)).

to request for a travel ticket and a hotel reservation during a specific period according to a specific objective (called mission). This request can be accepted or rejected by his/her hierarchical superior. In the case of an acceptance, the travel ticket and hotel room are booked by contacting a specific travel agency. The specification of this 'Travel' Web application is performed using a TEFSM model through IF formalism [6].

4.1 Functional Specification of Travel System

To perform this formal specification, we rely in our approach on one of the languages based on TEFSM: IF formalism. IF is usually used to model functional behavior of communicating systems such as network protocols, services and Web applications. The Intermediate Format (IF) language can be considered as a common representation model for other existing languages. It was originally developed to sit between languages as SDL, Promela or Lotos [5] but it has been extended to tackle other notations, as UML [10].

The semantic of time in IF language is the same as in TEFSM. That is: (i) a timed behavior of a system can be controlled through clocks (or timers). (ii) The time progress in some states before executing the selected transitions. (iii) Transitions take zero time to be executed (instantaneous transitions). The specification of 'Travel' system using IF language provides the metrics presented in Table 3.

Processes	States	Transitions	Signals	Variables
basic_traveler	5	9	10	7
travel	2	7	7	8
traveler_mission	7	12	11	8
travel_mission	7	11	14	6
validator_mission	4	6	5	5

Table 3. IF Travel System Specification

4.2 Security requirements for Travel System

France Telecom proposed a preliminary version of the case study Travel, in which some informal security requirements are provided. Basing on these requirements, we formally specified later a set of 34 security rules using Nomad language. Several of these rules are time-related. For matter of space, we only present in this paper two of them:

- Rule 1. The following prohibition rule expresses that 2 mission requests of the same traveler must be separated by at least 2 minutes:

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

- Rule 2. The following permission rule expresses that a traveler can request for another travel propositions list within a delay of 10 minutes after the last possible request:

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

- Rule 3. The following 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 another request to remind the potential mission validator. This reminder is sent within a delay of (10080 min = 7 days):

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

4.3 Rules Integration

The integration of the security rules was performed based on the methodology described in sections 3. This integration allowed to add several clocks, transitions, processes and to restrain several transitions. The following table 4 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).

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

Table 4. IF Travel System Modifications According to Each Rule

5 Conclusion and Future Work

In this paper, we presented a framework for integrating timed security policies specified in Nomad within a functional specification of the system described in Timed Extended Finite State Machine model. We proposed several algorithms to automate this integration process. Then, we applied our framework to a representative industrial case-study provided by France Telecom. We showed that our approach allows the specification of various modalities such as obligation, permission and prohibition with timed constraints and makes it possible to obtain a secure system specification. Notice that our integration approach dealing with timed security rules is original since no previous work dealt with it before. Furthermore, the obtained secure system can be useful for several purposes such as code generation, specification correctness proof, model checking and automatic test generation.

As future work, we will use the formal secure specification of Travel system to derive automatically test cases to validate its implementation. The automatic test generation will target security issues. This task will be performed using `TestGen-IF` tool [8, 20] developed in our laboratory. `TestGen-IF` is based on Hit-or-Jump [7] algorithm that is especially used for components testing to perform test sequences generation from IF system specifications.

References

[1] I. 9646-1. *Information Technology - Open Systems Interconnection - Conformance testing methodology and framework Part 1: General Concepts*.

[2] 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 *Policy'03*, June 2003.

[3] V. S. Alagar and G. Ramanathan. Functional Specification and Proof of Correctness for Time Dependent Behaviour of Reactive Systems. *Formal Asp. Comput.*, 3(3):253–283, 1997.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[5] M. Bozga. *Symbolic Verification for Communication Protocols*. PhD thesis, VERIMAG/IMAG, 1999.

[6] M. Bozga, S. Graf, L. Mounier, and I. Ober. IF Validation Environment Tutorial. In *SPIN*, pages 306–307, 2004.

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

[8] A. R. Cavalli, E. M. de 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.

[9] A. R. Cavalli, J. P. Favreau, and M. Phalippou. Formal Methods for Conformance Testing: Results and Perspectives. In O. Rafiq, editor, *Protocol Test Systems*, volume C-19 of *IFIP Transactions*, pages 3–17. North-Holland, 1993.

[10] P. Chevalley and P. Thévenod-Fosse. Automated Generation of Statistical Test Cases from UML State Diagrams. In *COMPSAC*, pages 205–214, 2001.

[11] E. M. Clarke and D. Kroening. Tutorial: Software Model Checking. In J. Davies, W. Schulte, and M. Barnett, editors, *ICFEM*, volume 3308 of *Lecture Notes in Computer Science*, pages 9–10. Springer, 2004.

[12] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel. High Level Conflict Management Strategies in Advanced Access Control Models. *Electr. Notes Theor. Comput. Sci.*, 186:3–26, 2007.

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

[14] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[15] K. Li, L. Mounier, and R. Groz. Test Generation from Security Policies Specified in Or-BAC. In *COMPSAC (2)*, pages 255–260. IEEE Computer Society, 2007.

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

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

[18] V. K. Paleri. Automatic Generation of Code Optimizers from Formal Specifications. In *The Compiler Design Handbook*, pages 61–97. 2002.

[19] J.-C. Park and R. E. Miller. A Compositional Approach for Designing Multifunction Time-Dependent Protocols. In *ICNP*, pages 105–112. IEEE Computer Society, 1997.

[20] E. R. Vieira and A. Cavalli. Toward Test Suite Automatic Generation with Delayable Transitions and Timing-Fault Detection. In *RTCSA*, 2007.