Enhancing Formal Specification and Verification of Temporal Constraints in Business Processes

Saoussen Cheikhrouhou, Slim Kallel
ReDCAD Laboratory
Univ. of Sfax, Tunisia
{saoussen.cheikhrouhou,slim.kallel}@redcad.org

Nawal Guermouche
CNRS-LAAS, Toulouse, France
Univ. of Toulouse, INSA, Toulouse, France
nawal.guermouche@laas.fr

Mohamed Jmaiel
ReDCAD Laboratory
Univ. of Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

***Abstract*—Formal specification and verification support of time-related constraints constitute fundamental challenges for any Business Process Management (BPM) system. Reluctantly, the literature on the subject of formal specification and verification of advanced temporal constraints such as *absolute temporal* constraints associated with *relative temporal constraints* is scarce. In this paper, we propose a novel approach enabling the formal specification and verification of advanced temporal constraints of business processes. The particularity of our approach is that it caters for relative and absolute related temporal constraints while relying on the dependencies that can exist between theses constraints. In fact, it is important to deal with such dependencies to handle the violations that can arise as soon as possible at design step. To do so, we propose a formal approach which relies on the timed automata formalism. In this context, we propose a set of mapping rules and algorithms where the semantic of timed automata is preserved even if we deal with absolute and relative temporal constraints. Using the defined formal model, we investigate a model checking based verification process that aims at validating business processes against their absolute and relative temporal constraints.**

***Index Terms*—Temporal constraints; BPM; Formal specification; Formal verification**

I. Introduction

Business globalization highly urges collaborations among organisations with complementary skills to form an Inter-Organizational Business Process (IOBP). In this context, business processes should adhere to a wide range of temporal requirements which rise from legal, regulatory, and managerial rules. For the collaboration of such business processes, it is necessary that *temporal constraints* of the model can be verified. Failing to manage advanced temporal constraints in business processes turns out in higher process execution costs. Although time dimension has been highly addressed in the literature [1], [2], [3], [4], [5], temporal constraints need to be viewed from multiple perspectives namely, the *relative* and *absolute* temporal constraints.

A relative temporal constraint refers to a constraint using relative time, i.e., with measurements of the time that passes between two observable events. An absolute temporal constraint refers to a constraint using time stamped in absolute time, measured from a global time clock which is never reset [6]. Generally, temporal constraints and especially absolute temporal constraints are used at run-time for monitoring purposes allowing thus for efficient time management and avoiding temporal failures [7], [8]. Nevertheless, before the execution stage, detecting and correcting temporal violations of the model as early as possible (i.e. from the specification stage), in order to guarantee the trustiness of process models has multiple of benefits. Indeed, it has the potential for avoiding considerable loss in time, human resources and revenue of the organisation. Even though many research approaches [1], [2], [3], [8], [4], [9], [5] have tackled the problem of temporal constraints specification and verification at process design time, they do not consider advanced temporal constraints. Furthermore, most of them stick to a strong assumption that temporal constraints of the model are always relative. This assumption is too restrictive.

Only few works [6], [8] have focused on the formal specification of absolute temporal constraints in business process models. But, these works do not propose reasoning mechanisms to handle the problem of formal verification. However, in most real scenarios, the definition of automatic formal verification of expressive timed business processes specification is crucial. This, in turn, has led to an increasing demand for innovative mechanisms and technologies that support the time management in the process lifecycle.

In this paper, our goal is to handle the problem of automatic verification of BPMN process models [10], [11] where we cater for advanced temporal constraints which rely on *relative* and *absolute* time and their dependencies. To do so, we first propose a formal model based on timed automata. Based on the defined formal model, we propose a model checking based mechanism to enable the formal verification we intend. It is worth noting that the particularity of our proposed approach is to enable absolute temporal constraints associated with relative temporal constraints of business processes to be modeled as timed automata. This is made without hampering their efficient analysis and mainly permitting their later verification to cater for these constraints and the dependencies that can exist through existing model checking tools. At the best of our knowledge, this is the first work that shows how to model and verify absolute temporal constraints associated to relative temporal constrains and which allows to define a relationship between them using the formalism of timed automata.

This paper is organized as follows. Section II provides an overview of the proposed approach. Section III presents the different temporal constraints we consider in this paper.

Section IV outlines the proposed mapping of timed proecesses into timed automata. Section V outlines the proposed formal verification approach to detect the temporal violations of process models. A review of related literature is given in Section VI. Finally, Section VII concludes.

II. The proposed approah in a nutshell

Figure 1 gives an overview of our approach which consists of four steps.

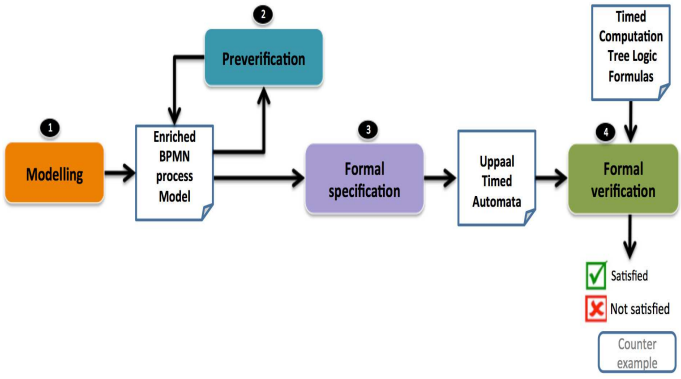


Fig. 1. Overview of the proposed approach

Firstly, the *modelling* step depicted by label ❶ in Fig. 1, enables the designer to specify the temporal constraints of the process using the defacto industrial standard for process modelling, BPMN.

Second, based on these extensions, absolute temporal constraints of the process are the unique concern of the *preverification* step depicted by label ❷ in Fig. 1. This latter helps the designer to uncover subtle mistakes in specification, early on, even before conducting the formal verification step. A set of rules have been proposed to prevent the designer to specify some faulty temporal combinations of constraints. An example of such faulty constraints can be :

1) *A given activity, say A_1, must finish its execution on 10 AM, while a successive activity following it, say A_2 must start on 8 AM.*
2) *A given activity, say A_1, must finish its execution no earlier than 10 AM, while a successive activity following it, say A_2 must start no later than 9 AM .*

The aim behind the preverification step is to enable, when possible, a direct mapping of timed processes to valid timed automata (i.e. with positive clock values). Space limitations prevent a detailed exhibition of the *modelling* ❶ and the *preverification* ❷ steps. Further details can be found in [12].

Next, enriched process models are mapped into timed automata with the help of the *formal specification* step (depicted by label ❸ in Fig. 1). The particularty of this latter is to enable a large set of absolute temporal constraints of processes to be modeled as timed automata.

Finally, the fourth step, *formal verification* (depicted by label ❹ in Fig. 1), provides a model checking based verification process to detect possible temporal violations enabling thus to react to them predictively. The verification process outlined in this paper goes far beyond the simple verification of the structural properties of the model (eg. deadlock). Precisely, it enables the verification of user-defined absolute and relative temporal constraints. The *formal specification* ❸ and *formal verification* ❹ steps are the focus of this paper.

III. Timed business process specification

Before explaining the steps of the formal specification of temporal constraints in business processes, we consider some assumptions and introduce some definitions.

Assumption: We assume a structured representation of process models. The fact that it is possible to represent unstructured models in the BPMN notation does not limit the scope of our work. Indeed, as shown in [13], most of unstructured process models can be automatically translated into structured ones.

In essence, a process model is represented as a tree whose leaves represent activities and whose internal nodes represent either events (eg. Start Event SE) or gateways (eg. sequence (SEQ), parallel (PAR)). We formally capture the structured process models as follows.

Definition 1. *(Process Graph)*
Let Γ be a set of types of nodes. A Process Graph P is a tuple (N,E,τ,γ), in which:
-N is the set of nodes;
-$E \subseteq N * N$ is the set of edges; and
- τ : N$\rightarrow$ Γ is a function that maps nodes to their types
- γ is the set of temporal constraints labels of the process.

Note : $\gamma_{Rel}(N_i)$ (resp. $\gamma_{Abs}(N_i)$) denotes the relative (resp. absolute) temporal constraints of the node N_i.
Actually, Γ supports the following types of nodes : activities (Activity), events (i.e. Start Event(SE) and End Event(EE)) and gateways (i.e. sequence(SEQ), parallel(PAR), inclusive(INCL) and exclusive(EXCL)). Hereafter, we present the temporal constraints we consider. Particularly, we present *relative* then *absolute temporal constraints*.

- Relative temporal constraints: We consider *intra-activity* and *inter-activity* temporal constraints. Intra-activity temporal constraints are those associated to one activity within the process model, such as:

- *Duration* : a given activity has minimum and maximum execution times.
- *Temporal Constraint Over Cardinality (TCOC)* : a given activity can be executed successively at most N times whithin a time period T.

The duration constraint is defined as follows. Let s(A) (resp. e(A)) be the starting (resp. the ending time) of the activity A. Let $MinA$ and $MaxA$ be two relative time values representing respectively the minimum and maximum durations of an activity A. The *Duration* constraint *Duration(A,$MinA$,$MaxA$)* is defined as :

$$MinA \leq \text{e}(A) - \text{s}(A) \leq MaxA$$

Inter-activity temporal constraints are constraints crossing the boundary of an activity in the process model, such as:

- Start-to-Finish (SF): A_2 can not finish until A_1 has started within a given time interval
- Start-to-Start (SS) : A_2 can not begin before A_1 starts within a time interval
- Finish-to-Start (FS) : A_2 can not begin before A_1 ends within a time interval
- Finish-to-Finish (FF) : A_2 can not finish until A_1 has finished within a time interval.

The Finish-to-Start Temporal Dependency constraint of two activities A_1, A_2, denoted *TD(*FS,A_1,A_2,$MinD$,$MaxD$*)*, is defined as :

$$MinD \leq \text{s}(A_2) - \text{e}(A_1) \leq MaxD$$

The latter definition denotes that the activity A_2 should starts its execution no later than $MaxD$ time units and no earlier than $MinD$ time units after the activity A_1 ends. For more details on the different temporal constraints we consider, we refer the reader to [11].

- Absolute temporal constraints*:* We now turn our attention to the specification of absolute temporal constraints of the process model. The succeeding listing summarizes the set of absolute temporal constraints that we consider in our specification approach. They are intra-activity temporal constraints focusing particularly on controling the start and finish times of process activities. We refer the reader interested in the semantics of the following constraints to [11]:

- Must Start On (MSO)/ Must Finish On (MFO)
- Start No Earlier Than (SNET)/ Finish No Earlier Than (FNET)
- Start No Later Than (SNLT)/ Finish No Later Than (FNLT)

An absolute temporal constraint is formally denoted as $AbsTC(N, t, T, v)$ where N is the concerned node by the temporal constraint of type *t* (S: for start or E: for end) and T (ON: for on, NET: for not earlier than or NLT: not later than) and of value v. $AbsTC(A, S, ON, 8)$ to denote that *A Must Start On 8 AM/ PM*.

IV. Timed formal model: Mapping of timed business process specification

After presenting the different temporal constraints we consider, in this section we present how we map timed business processes into timed automata. Indeed, our goal is to verify and to detect temporal violations that can arise. A timed automaton is an automaton where transitions are labelled by an alphabet and temporal constraints, called guards , and resets of clocks. The former represent simple conditions over clocks, and the latter are used to reset values of certain clocks to zero. The guards specify that a transition can be fired only if the corresponding guards are satisfiable.

A temporal constraint is a conjunction of expressions that compares the value of a clock $x \in X$, to a positive real constant $a \in \mathbb{R}_+$. Let X be a set of clocks. The set of *constraints* over X, denoted $\Psi(X)$, is defined as follows:

true $\mid x \bowtie a \mid \psi_1 \wedge \psi_2$, where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, $\psi_1, \psi_2 \in \Psi(X)$ and $a \in \mathbb{R}_+$. Clock constraints will be used as guards and invariants for timed automata.

Definition 2. *(Timed Automata)*
Recall [14] that a timed automaton *(or TA for short) is a tuple $\mathcal{A} = (L, X, l_0, A, T, Inv)$, where L is a finite set of locations (or nodes), X is a finite set of clocks, $l_0 \in L$ is an initial location, A is a finite alphabet, standing for actions, $T \subseteq L \times A \times \Psi(X) \times 2^X \times L$ is a set of transitions between locations with an action, a guard and a set of clocks to be reset, and $Inv : L \rightarrow \Psi(X)$ assigns invariants to locations.*

We notice that we use the syntax and semantic of timed automata as used in the model checker Uppaal. Given t = $(l, \alpha, \psi, r, l') \in T$, l is the source location, α is the label; ψ is the guard; r is the set of clocks to reset and l' is the target location. We use L^u to denote the subset of urgent locations in L ($L^u \subseteq L$). An urgent location is a location where no delay is allowed.

For more readability, we shall write $l \xrightarrow{\alpha,\psi,r} l'$ when $(l, \alpha, \psi, r, l') \in T$. To note transitions, we write only non-empty sets (i.e. $l \xrightarrow{\emptyset,\emptyset,\emptyset} l'$ is denoted $l \rightarrow l'$).
As in verification tools, we assume that invariants are either true or conjunctions of simple constraints of the form $x \bowtie a$ where $\bowtie \in \{\leq, <\}$, $x \in X$ and $a \in \mathbb{N}$.

Definition 3. *(Union of Timed Automata)*
The union of two tuples is defined as $(L, X, l_0, A, T, Inv) \uplus (L', X', l'_0, A', T', Inv') = (L'', X'', l''_0, A'', T'', Inv'')$ with $L'' = L \cup L'$,$X'' = X \cup X'$,$l''_0 = l_0 \cup l'_0$,$A'' = A \cup A'$,$T'' = T \cup T'$, $Inv'' = Inv \cup Inv'$, and $L''^u = L^u \cup L'^u$.

A. Mapping of relative temporal constraints

Fig. 2 depicts the mapping that we propose from BPMN tasks, events, and gateways to timed automata. Fig. 2(a) shows the mapping of the BPMN start and end events.

The automatic mapping of these constructs is ensured by the help of the *Create_Automaton* function. *Create_Automaton* is an elementary function that, according to its input parameteres, creates the suitable timed automaton. Given an activity A, which is provided with a duration temporal constraint *Duration(*$A, MinA, MaxA$*)*, the output of the *Create_Automaton* function is a timed automaton $\mathcal{A} = (L, X, l_0, A, T, Inv)$ (see Fig. 2(c)) such as :

L = $\{A_{Ready}, A_{Working}, A_{Finish}\}$
X = $\{x\}$
L^u = $\{A_{Ready}, A_{Finish}\}$
$Inv(A_{Working}) = x \leq MaxA$
T = $\{A_{Ready} \xrightarrow{\emptyset,\emptyset,x} A_{Working}, A_{Working} \xrightarrow{\emptyset,x \geq MinA \wedge x \leq MaxA,\emptyset} A_{Finish}\}$

If additional relative temporal constraints such as temporal constraint over cardinality (TCOC) should be considered (see Fig. 2(d)), the *Create_Automaton* returns a timed automaton $\mathcal{A}$ as follows :

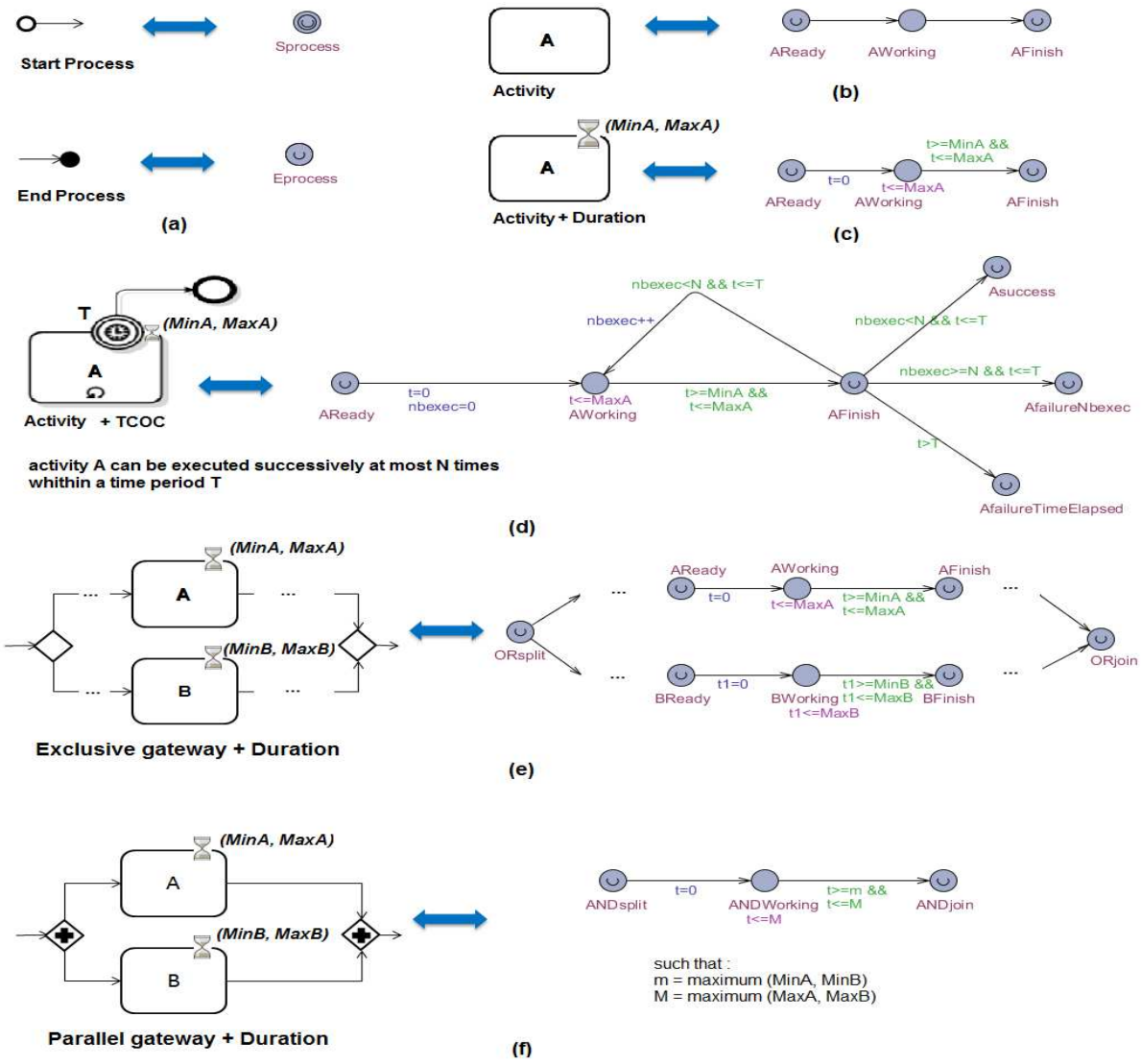


Fig. 2. Mapping of BPMN into timed automata

$L = L \cup \{A_{Ready}, A_{Working}, A_{Finish}, A_{Success}, A_{failureNbexec}, A_{failureTimeElapsed}\}$
$X = X \cup \{x\}$
$L^u = L^u \cup \{A_{Ready}, A_{Finish}, A_{Success}, A_{failureNbexec}, A_{failureTimeElapsed}\}$
$Inv(A_{Working}) = x \leq MaxA$
$T = \{A_{Ready} \xrightarrow{\emptyset,\emptyset,\{x,nbexec\}} A_{Working},$
$A_{Working} \xrightarrow{\emptyset,x \geq MinA \wedge x \leq MaxA,\emptyset} A_{Finish},$
$A_{Finish} \xrightarrow{nbexec++,x \leq T \wedge nbexec \leq N,\emptyset} A_{Working},$
$A_{Finish} \xrightarrow{\emptyset,x \leq T \wedge nbexec \geq N,\emptyset} A_{Success},$
$A_{Finish} \xrightarrow{\emptyset,x \leq T \wedge nbexec \geq N,\emptyset} A_{failureNbexec},$
$A_{Finish} \xrightarrow{\emptyset,x \geq T,\emptyset} A_{failureTimeElapsed}\}$

Due to space limitations, we have confined the definition of the $Create_Automaton$ algorithm to [12]. Fig. 2 (e) (resp. (f)) outlines the proposed mapping of exclusive (resp. parallel) gateways.

Algorithm 1; the *From Timed Process to Timed Automata* algorithm; is the main function of the *formal specification* step. The main goal of this function, as its name suggests, is to return a timed automaton from a timed process graph. As local variables of this function, we notice the $l_{current}$ and Ref. The variable Ref is used to save the system state.

$l_{current} \in L$ is a variable of type location used to distinguish the location used to ensure the union of automata. Obviously, the location needed for the transition linking two tuples must be precised each time. At first, the algorithm populates the set of urgent locations by two locations, namely the $S_{Process}$ and $E_{Process}$. $S_{Process}$ is the initial location. Afterward, $l_{current}$ is initialized to $S_{Process}$. The main function calls the *Spec_Rel_Abs_TC* procedure given by the Algorithm 2, which constructs the timed automaton of the whole process considering both its relative and absolute temporal constraints.

Algorithm 1 From Timed Process to Timed Automata

1: **function** FROM_TIMED_PROCESS_TO_TIMED_AUTOMATA
2: **Input** $P(N, E, \tau, \gamma)$
3: **Output** $\mathcal{A} = (L^u \cup L^c, X, l_0, A, T, Inv)$
4: **local** $l_{current}$,Ref
5: $L^u = \{S_{Process}, E_{Process}\}$
6: $l_{current} \leftarrow S_{Process}$
7: $l_0 \leftarrow S_{Process}$
8: $Ref(\emptyset, \emptyset, \emptyset)$
9: *Reach the first node of the process graph* N_i
10: *Spec_Rel_Abs_TC* (N_i,P,$\mathcal{A}$,$l_{current}$,Ref)
11: $T = T \cup l_{current} \rightarrow E_{Process}$
12: **end function**

Finally, the first location of the constructed automaton will be linked to the $S_{Process}$ location and its last location will be respectively linked to the $E_{Process}$ location.

In the *Spec_Rel_Abs_TC* procedure, we differenciate between four major parts of this algorithm. The first one (lines 5-8) is devoted to nodes which are in a sequential process flow. The second part (lines 9-18) deals with nodes in exclusive gateways. The third part (lines 19-25) is dedicated to nodes in parallel or inclusive gateways. In this latter, the $Max_Duration$ and $Min_Duration$ functions are used to calculate the minimum and maximum durations of the gateway while considering the delays caused by relative as well as absolute temporal constraints.

Finally, the fourth part (lines 26-44) deals with activity nodes and resorts to the $Create_Automaton$ function to create the appropriate automaton depending on the intra-activity relative temporal constraints set of the activity (i.e. Duration or TCOC). Throughout the execution of algorithm 2, each time, it calls the $Create_Automaton$ function, a new automaton is generated. Next, the algorithm ensures the union of the automata and draws a new transition linking the location $l_{current}$ to the corresponding newly-created location depending on the output of the $Create_Automaton$ function. Obviously, the $l_{current}$ is updated accordingly.

Lines 35-37 focus on the formal specification of *temporal dependencies* (i.e. SS/SF/FS/FF) which are inter-activity relative temporal constraints. Indeed, it is the aim of the called $Spec_Rel_TD$ procedure to check even the given activity node is a source or destination of the temporal dependency constraint in order to add whether the required update or guard for the appropriate location of the timed automaton. Due to space limitations, we have confined the definition of the $Spec_Rel_TD$ algorithm to [12]. The rest of algorithm 2 (lines 38-43) is detailed in the succeeding subsection.

B. Mapping of absolute temporal constraints

As said previously, the goal of our approach is to propose timed automata based formal specification to capture in addition to relative temporal constraints, absolute temporal constraints features and to enable their verification at design time. This is ensured by establishing clear relation dependencies between clocks of the process timed automaton. Obviously,

Algorithm 2 Specification Relative Absolute Temporal Constraints

1: **procedure** SPEC_REL_ABS_TC
2: **Input** N_i,$P(N, E, \tau, \gamma)$
3: **Input/Output** $\mathcal{A} = (L, X, l_0, A, T, Inv)$,$l_{current}$,$Ref$
4: **local** $\mathcal{A}' = (L', X', l'_0, A', T', Inv')$
5: **if** $\tau(N_i, P) = SEQ$ **then**
6: **for all** *node*N_j of the sequence flow **do**
7: *Spec_Rel_Abs_TC* ($N_j, P, \mathcal{A}$,$l_{current}$,Ref)
8: **end for**
9: **else if** $\tau(N_i, P) = EXCL$ **then**
10: $L^u = L^u \cup \{OR_{iSplit}, OR_{iJoin}\}$
11: $T = T \cup l_{current} \rightarrow OR_{iSplit}$
12: $l_{current} \leftarrow OR_{iSplit}$
13: $Ref' \leftarrow Ref$
14: **for all** *node*N_j of the exclusive gateway **do**
15: *Spec_Rel_Abs_TC* ($N_j, P, \mathcal{A}$,$l_{current}$,Ref')
16: $T = T \cup l_{current} \rightarrow OR_{iJoin}$
17: **end for**
18: $l_{current} \leftarrow OR_{iJoin}$
19: **else if** $\tau(N_i, P) \in (PAR, INCL)$ **then**
20: $MinD \leftarrow Min_Duration(N_i, P)$
21: $MaxD \leftarrow Max_Duration(N_i, P)$
22: $\mathcal{A}' = Create_Automaton(N_i, PAR, Duration(N_i, MinD, MaxD))$
23: $\mathcal{A} = \mathcal{A} \uplus \mathcal{A}'$
24: $T = T \cup l_{current} \rightarrow AND_{iSplit}$
25: $l_{current} \leftarrow AND_{iJoin}$
26: **else**/*$\tau(N_i, P) = Activity$ */
27: $\mathcal{A}' = Create_Automaton(N_i, Activity, \gamma_{Rel}(N_i))$
28: $\mathcal{A} = \mathcal{A} \uplus \mathcal{A}'$
29: $T = T \cup l_{current} \rightarrow A_{iReady}$
30: **if** *TCOC* is a temporal constraint for the node N_i **then**
31: $l_{current} \leftarrow A_{iSuccess}$
32: **else**
33: $l_{current} \leftarrow A_{iFinish}$
34: **end if**
35: **for all** Temporal Dependency TD_k of the node N_i **do**
36: Spec_Rel_TD(TD_k,N_i,P,$\mathcal{A}$)
37: **end for**
38: **for all** Absolute Temporal constraint $AbsTC_k$ of the node N_i **do**
39: **if** $Ref \neq (\emptyset, \emptyset, \emptyset)$ **then**
40: Spec_Abs_TC($AbsTC_k$,N_i,Ref)
41: **end if**
42: Update_Ref($AbsTC_k$,Ref)
43: **end for**
44: **end if**
45: **end procedure**

if we consider the two following temporal constraints : *A given activity, say* A_1*, must finish its execution on 5 AM (i.e.,* $(AbsTC(A_1, F, ON, 5))$ *), while a successive activity following it, say* A_2 *must start on 8 AM (i.e.,* $(AbsTC(A_2, S, ON, 8))$. Our approach considers the first absolute temporal constraint $(AbsTC(A_1, F, ON, 5))$ as a *timed reference* to subsequently specify that exactly 3 hours must separate the starting time of A_2 and the finishing time of A_1. In other words, we capture the relation between absolute temporal constraints using relative clocks.

Since dependencies can exist between a set of absolute

temporal constraints, and in order to capture all dependencies, we explore combinations of absolute constraints pairs. We establish, a relation between each absolute temporal constraint with the identified timed reference.

In the following, we show the details of how we map absolute temporal constraints using timed automata. We note that a reference vector $Ref(ON_{Ref}, NET_{Ref}, NLT_{Ref})$ consists of three timed references as follows ON_{Ref}, NET_{Ref} and NLT_{Ref}, where every reference is an absolute temporal constraint. This latter is the focus of lines (38-43) of algorithm 2 in which the update of the reference vector is made by the help of the $Update_Ref$ procedure. A detailed version exhibiting the $Update_Ref$ function can be found in [12].

We consider now the $Spec_Abs_TC$ procedure presented in Algorithm 3, which ensures the specification of absolute temporal constraints of the model.

Given an absolute temporal constraint $AbsTC_i(A_i, t_i, T_i, v_i)$, and depending on the constraint type T_i, lines (4-22) are devoted to the MSO/ MFO constraints (i.e. $T_i = ON$), lines (22-37) deal with the SNET/ FNET constraints (i.e. $T_i = NET$) and finally, lines (38-50) are dedicated to the SNLT/ FNLT constraints (i.e. $T_i = NLT$). Each time, the $Spec_Abs_TC$ procedure, computes the required updates and guards needed for the enrichement of the timed automaton. Furthermore, it calls the add_Update and add_Guard functions, which, according to the Ref vector, place the updates and guards properly on the automaton. The details of the add_Update and add_Guard functions are omitted to facilitate the readability of the algorithm. For more details, we refer the reader to [12].

We note that, there are some cases where our approach can not add details to timed automata. Our algorithms do not add any details for the specification of some constraints if :

- The succession of such constraints can never cause violations. For instance, *a given activity, say A_1, must finish its execution on 10 AM, while a successive activity following it, say A_2 must start no earlier than 8 AM.* This is implicitly ensured by the structure of the process which oblige that A2 follows in the sequence flow the activity A_1.
- Or, even though the succession of such constraints can cause violations, our approach is unable to help the designer by adding the needed specification details. Indeed, we are unable to establish a relation (in the temporal space) between activities with illimited starting/ firing time (i.e. tends to $-\infty$ or $+\infty$). For example, *a given activity, say A_1, must start its execution no earlier than 5 AM, while a successive activity following it, say A_2 must start its execution no earlier than 8 AM.*

Example 1. *(A purchase order process mapping)*
For illustration purposes, we consider particularly an excerpt from the BPMN diagram of the purchase order process in a manufacturing organisation depicted in Fig.3.

The process is triggered when a customer submits a purchase order (Receive order). Then, the organisation checks whether the ordered articles are available or not (Check availability). Following that, the customer is asked for financial settlement (Receive settlement) and the goods are subsequently delivered (Deliver goods). By the reception of the order, the process meets its end. Fig.3 exhibit moreover,

Algorithm 3 Adding Absolute Temporal Constraints to the Specification

1: **procedure** SPEC_ABS_TC
2: **Input** $AbsTC_i(A_i, t_i, T_i, v_i)$, $Ref(ON_{Ref}, NET_{Ref}, NLT_{Ref})$ /*$Ref(ON_{Ref}(N_{ON}, t_{ON}, T_{ON}, v_{ON}), \quad NET_{Ref}(N_{NET}, t_{NET}, T_{NET}, v_{NET}), NET_{Ref}(N_{NLT}, t_{NLT}, T_{NLT}, v_{NLT}))$*/
3: **Input/Output** $\mathcal{A} = (L, X, l_0, A, T, Inv)$
4: **if** $T_i = ON$ **then**
5: **if** $ON_{Ref} \neq \emptyset$ **then**
6: $X = X \cup \{x_{ON_i}\}$
7: add_Update(x_{ON_i}, N_{ON},t_{ON})
8: $v = v_i - v_{ON}$
9: add_Guard(x_{ON_i}, A_i, t_i, v)
10: **else if** $NET_{Ref} \neq \emptyset$ **then**
11: $X = X \cup \{x_{NET_i}\}$
12: add_Update($x_{NET_i}, N_{NET}, t_{NET}$)
13: $v = v_i - v_{NET}$
14: add_Guard(x_{NET_i}, A_i, t_i, v)
15: **else if** $NLT_{Ref} \neq \emptyset$ **then**
16: **if** $v_i \geq v_{NLT}$ **then**
17: $X = X \cup \{x_{NLT_i}\}$
18: add_Update($x_{NLT_i}, N_{NLT}, t_{NLT}$)
19: $v = v_i - v_{NLT}$
20: add_Guard(x_{NLT_i}, A_i, t_i, v)
21: **end if**
22: **end if**
23: **else if** $T_i = NET$ **then**
24: **if** $ON_{Ref} \neq \emptyset$ **then**
25: **if** $v_i \geq v_{ON}$ **then**
26: $X = X \cup \{x_{ON_i}\}$
27: add_Update(x_{ON_i}, N_{ON}, t_{ON})
28: $v = v_i - v_{ON}$
29: add_Guard(x_{ON_i}, N_{ON}, t_{ON})
30: **else if** $NLT_{Ref} \neq \emptyset$ **then**
31: **if** $v_i \geq v_{NLT}$ **then**
32: $X = X \cup \{x_{NLT_i}\}$
33: add_Update($x_{NLT_i}, N_{NLT}, t_{NLT}$)
34: $v = v_i - v_{NLT}$
35: add_Guard(x_{NLT_i}, A_i, t_i, v)
36: **end if**
37: **end if**
38: **else**/'*$T_i = NLT$ */
39: **if** $ON_{Ref} \neq \emptyset$ **then**
40: $X = X \cup \{x_{ON_i}\}$
41: add_Update(x_{ON_i}, N_{ON}, t_{ON})
42: $v = v_i - v_{ON}$
43: add_Guard(x_{ON_i}, A_i, t_i, v)
44: **else if** $NET_{Ref} \neq \emptyset$ **then**
45: $X = X \cup \{x_{NET_i}\}$
46: add_Update($x_{NET_i}, N_{NET}, t_{NET}$)
47: $v = v_i - v_{NET}$
48: add_Guard(x_{NET_i}, A_i, t_i, v)
49: **end if**
50: **end if**
51: **end if**
52: **end procedure**

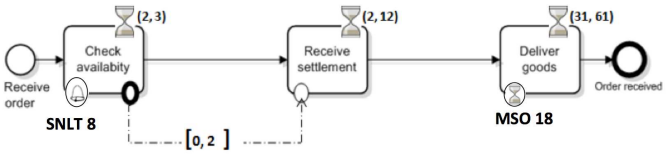


Fig. 3. The purchase order process enriched with temporal constraints

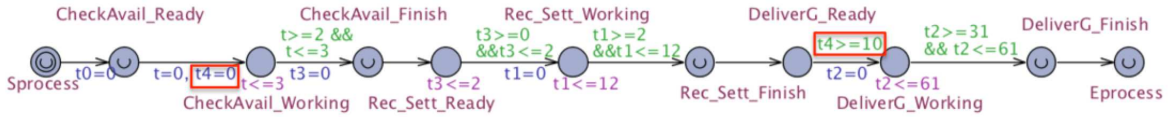


Fig. 4. Mapping of the purchase order process into Timed Automata

the relative temporal constraints of the process model. We can notice that each activity of the process is provided with a Duration temporal constraints. For instance, Duration(Deliver goods,$MinD$,$MaxD$) with $MinD$= $31hours$ and $MaxD$= $61hours$, is the selected notation to precise the minimum and maximum execution times of the activity Deliver goods. Furthermore, the process is provided by a temporal dependency Finish to Start (FS) as follows:
TD(FS,Check availability,Receive settlement,$MinD$,$MaxD$) with $MinD = 0hours$ and $MaxD = 2$ hours. Further absolute temporal constraints of the model are depicted to precise that the activity Check availability has to start no later than 8 H (SNLT 8) and that the activity Deliver goods must start exactly on 18 H (MSO 18).
Figure 4 depicts the timed automata resulting from the application of our main algorithm (From Timed Process to Timed Automata) to the process 3. The Spec_Rel_Abs_TC procedure detects that this is a sequential process flow. Afterward, it will be executed recursively until processing the activity Deliver goods. Subsequently, the algorithm Create_Automaton is run thrice (once for each activity of the process model). It is hence the role of the Spec_Rel_Abs_TC procedure to link each time the newly constructed automaton (i.e. the output of the Create_Automaton function) to the main automaton. The FS temporal dependency of the process model is tackled by the $Spec_Rel_TD$ procedure which adds the update ($t3 = 0$) and the guard ($t3 \geq 0$ && $t3 \leq 2$) in the appropriate place. We deal now with the two absolute temporal constraints of the model. Initially, all references are set to the emptyset $Ref(\emptyset, \emptyset, \emptyset)$. With the meeting of the first absolute constraint of the process model, (i.e. the first SNLT 8), the value NET_{Ref} is settled by the help of the $Update_Ref$ procedure. With the meeting of the second absolute constraint of the process model (i.e. the MSO 18), the $Spec_Abs_TC$ procedure adds the update ($t4 = 0$) for the appropriate activity automaton according to the reference. Here the update will concern the activity Check availability. And it will be placed exactly between the CheckAvail_Ready and the CheckAvail_Working locations since this is a start temporal constraint (i.e. SNLT 8 and not FNLT 8). Consequently, the $Spec_Abs_TC$ procedure calculates the real bounds of the guard (i.e. 18 - 8 = 10 hours) in order to affect ($t4 \geq 10$) in the appropriate place in the automaton (see Fig. 4).

V. Formal Checking

The definition of temporal constraints allows to specify constrained process models that may encounter a deadlock situation due to inconsistencies between nested temporal constraints. Our approach allows the verification of deadlock freedom. Moreover, our work goes far beyond the simple verification of the structural properties of the process (eg. deadlock). Precisely, we ensure the verification of user-defined temporal constraints such as deadlines. For instance, the designer can verify delays between two activities A_1 and A_2 of a process or between the start of the process and its end. In this context, we use the real-time model checker UPPAAL for the formal verification of timed processes. The UPPAAL model checker ensures the verification of systems, modelled as timed automata (TA) against a desired set of properties defined using a rich subset of CTL (Computation Tree Logic) formulas. Given the generated timed automata (see Fig.4) corresponding to business processes described in example1 (see Fig.3) , we propose to verify the following CTL properties:

A[] not deadlock: to ensure deadlock freeness of the process,

A[] (Process.Eprocess imply $t_0 \leq 100$**)**: to verify the process deadline is met.

Afterward, both timed automata models and queries for temporal constraints are input into the UPPAAL model checking engine.

Example 2. *In this example, we first handle the verification of the purchase order process without considering its absolute temporal constraints (without considering the guards and updates relating to clock t_4 of Fig.4).*

The verification results show that the corresponding process model is deadlock free and meets the deadline (i.e. both CTL properties are verified).
To show the value of our approach, let's consider now the automaton enriched with absolute temporal constraints (while considering the guards and updates relating to clock t4 of Fig.4). While verifying the same CTL properties, UPPAAL reports that the first property is satisfied (i.e. the process is deadlock free), but the second property is not (i.e. the process does not meet the deadline). The returned results from UPPAAL shows the usefulness of our solution. Indeed, it helps business analysts to detect temporal violations resulting from absolute temporal constraints at process design time. It is clear that manually considering the combination of absolute temporal constraints of the example while respecting the process deadline is a fastidious and error prone task. Failing to manage absolute temporal constraints in processes turns out in higher process execution costs, either by loss of productivity or lack of coordination. Indeed, it is difficult to detect their inconsistencies, especially in complicated business processes. Our approach enables the verification of these constraints exhaustively and effectively by using model checking techniques.

VI. Related work

Modeling and managing temporal constraints in business processes has long been a topic of intensive researches [1], [2], [3], [8], [4], [9], [5].

The work presented in [9] proposes a formal specification of BPMN [10] with timed automata. First, the authors extend BPMN to handle temporal constraints as the minimum and maximum execution times of activities. Second, they provide an automatic mapping of the extended BPMN into timed automata. CTL formulas are used to verify some features, such as deadlock and bottlenecks. The scope of this paper is limited to a small subset of BPMN elements. Additionnally, this BPMN extension permits to specify temporal constraints related to only one activity within the process model and does not consider timed properties related to a set of activities, such as inter-activity temporal constraints.

In [2], the author uses temporal properties in order to analyze the timed compatibility in Web service composition. A formal model based on timed automata is proposed. The UPAAL model checker was used to detect some structural problems due to temporal conflicts. The clock ordering process is used to verify deadlock freeness due to time constraints conflicts. Nevertheless, the scope of this paper is limited to the verification of time constraints only caused by message interaction between services of the process.

Regarding the concept of absolute temporal constraints specification, two approaches [6], [8] may be relevant.

The approach proposed in [8] covers the specification of temporal constraints for the web service domain using a new proposed language, XTUS-Automata. This latter combines timed automata (TA) and extends time unit system (XTUS) to allow specifying temporal properties involving relative time as well as absolute time. It is worth noting that this paper offers interesting specification patterns. Nevertheless, it is restricted to the formal verification of deadlock using the model checker UPAAL while considering only relative temporal constraints. The rich specification of absolute temporal constraints has no impact on the verification step.

The approach presented in [6] has accordingly attempted to address the need to model absolute temporal constraints on timed automata. Compared to our work, it does neither provide algorithms to automatically specify temporal constraints of the process nor consider rich absolute temporal constraints (Eg. MSO, SNET and FNLT).

As argued earlier, existing research approaches either targets a limited set of relative temporal constraints or, the absolute temporal constraints are only considered at specification step [11], [6].

VII. Conclusion

In this paper, we handled the verification problem of timed business processes. In this context, we have presented a novel model checking based verification approach. Particularly, we handled the problem of verification while considering advanced relative and absolute temporal constraints. To capture these constraints and their emerging dependencies, we proposed a mapping step whose aim is to map timed business processes into timed automata. Finally, we used the model checker UPPAAL to validate timed business processes and a set of requirements. Since business processes can be involved in collaborative environnements, it is worthwhile to handle temporal violations occuring during inter-business collaborations. We intend to further investigate on this problem to enhance the specification and verification of relative and absolute temporal constraints in concurrent processes sharing resources and exchanging messages.

Acknowledgment

Part of this work has been supported by FP7-ICT IMAGINE research and development project, co-funded by the European Commission under the Virtual Factories and Enterprises (FoF-ICT- 2011.7.3, Grant Agreement No: 285132).

References

[1] J. Eder and A. Tahamtan, "Temporal Conformance of Federated Choreographies," in *Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA)*, ser. LNCS, S. S. Bhowmick, J. Küng, and R. Wagner, Eds., vol. 5181. Springer, 2008, pp. 668–675.

[2] N. Guermouche and C. Godart, "Timed Conversational Protocol Based Approach for Web Services Analysis," in *Proceedings of the 8th International Conference on Service-Oriented Computing, ICSOC*, ser. LNCS, vol. 6470. Springer, 2010, pp. 603–611.

[3] R. Kazhamiakin, P. K. Pandya, and M. Pistore, "Representation, Verification, and Computation of Timed Properties in Web," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE Computer Society, 2006, pp. 497–504.

[4] Y. Du, P. Xiong, Y. Fan, and X. Li, "Dynamic Checking and Solution to Temporal Violations in Concurrent Workflow Processes," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 6, pp. 1166–1181, 2011.

[5] S. Cheikhrouhou, S. Kallel, N. Guermouche, and M. Jmaiel, "A Survey on Time-aware Business Process Modeling," in *Proceedings of the 15th International Conference on Enterprise Information Systems*. SCITEPRESS, 2013.

[6] H. Bohnenkamp and A. Belinfante, "Timed Testing with TorX," in *Proceedings of the FM 2005: Formal Methods*, ser. Lecture Notes in Computer Science, J. Fitzgerald, I. Hayes, and A. Tarlecki, Eds., vol. 3582. Springer, 2005, pp. 173–188.

[7] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "A Timed Extension of WSCoL," in *In the proceedings of the IEEE International Conference on Web Services, ICWS 2007.*, July 2007, pp. 663–670.

[8] S. Kallel, A. Charfi, T. Dinkelaker, M. Mezini, and M. Jmaiel, "Specifying and Monitoring Temporal Properties in Web Services Compositions," in *Proceedings of the 7th IEEE European Conference on Web Services*. IEEE Computer Society, 2009, pp. 148–157.

[9] K. Watahiki, F. Ishikawa, and K. Hiraishi, "Formal Verification of Business Processes with Temporal and Resource constraints," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2011, pp. 1173–1180.

[10] *Object Management Group (OMG), Business Process Model and Notation (BPMN), Version 2.0*, OMG Std., 2011.

[11] S. Cheikhrouhou, S. Kallel, N. Guermouche, and M. Jmaiel, "Toward a Time-centric modeling of Business Processes in BPMN 2.0," in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2013.

[12] S.Cheikhrouhou, S.Kallel, N.Guermouche, and M.Jmaiel, "Technical Report on the Verication of Temporal Constraints in Business Processes," http://www.redcad.org/PDFs/CheikhrouhouTR2014.pdf, Tech. Rep., 2014.

[13] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring Acyclic Process Models," in *Business Process Management*, ser. LNCS, vol. 6336. Springer, 2010, pp. 276–293.

[14] R. Alur, "Timed automata," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 1633. Springer, 1999.