1-1-2020

Business process specification, verification, and deployment in a mono-cloud, multi-edge context

Saoussen Cheikhrouhou
University of Sfax

Slim Kallel
University of Sfax

Ikbel Guidara
Université Claude Bernard Lyon 1

Zakaria Maamar
Zayed University

Computer Science and Information Systems 17(1):293–313 https://doi.org/10.2298/CSIS190430041C

Business Process Specification, Verification, and Deployment in a Mono-Cloud, Multi-Edge Context

Saoussen Cheikhrouhou[1], Slim Kallel[1], Ikbel Guidara[2], and Zakaria Maamar[3]

[1] ReDCAD
University of Sfax
Sfax, Tunisia
saoussen.cheikhrouhou@redcad.tn
slim.kallel@redcad.tn
[2] LIRIS
Claude Bernard Lyon 1 University
Lyon, France
ikbel.guidara@liris.cnrs.fr
[3] College of Technological Innovation
Zayed University
Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Abstract. Despite the prevalence of cloud and edge computing, ensuring the satisfaction of time-constrained business processes, remains challenging. Indeed, some cloud/edge-based resources might not be available when needed leading to delaying the execution of these processes' tasks and/or the transfer of these processes' data. This paper presents an approach for specifying, verifying, and deploying time-constrained business processes in a mono-cloud, multi-edge context. First, the specification and verification of processes happen at design-time and run-time to ensure that these processes' tasks and data are continuously placed in a way that would mitigate the violation of time constraints. This mitigation might require moving tasks and/or data from one host to another to reduce time latency, for example. A host could be either a cloud, an edge, or any. Finally, the deployment of processes using a real case-study allowed to confirm the benefits of the early specification and verification of these processes in mitigating time constraints violations.

Keywords: Business process, Cloud, Edge, Time constraint, Violation.

1. Introduction

Until recently cloud computing has been praised for both offering *elastic* (on-demand) resources and adopting *pay-as-you-go* model [20]. These 2 characteristics made cloud computing extremely popular among Information and Communication Technologies (ICT) practitioners who deployed software applications around the concept of Everything-as-a-Service (*aaS) that cloud computing embraces. Unfortunately, with the continuous advances in ICT and organizations' changing needs, cloud computing has shown some signs of "fatigue" when for instance, real-time applications call for almost zero time-latency. Transferring data to distant clouds is a potential source of delay and opens doors to unwanted interceptions. Luckily edge computing has been introduced to address some clouds' concerns like latency and security. According to Maamar et al. [16], edge and cloud are expected to work hand-in-hand and not compete.

In conjunction with embracing the latest ICT, all organizations capitalize on their Business Processes (BP) to remain competitive, cut costs, and sustain their growth. Referred to as *know-how*, *"...a business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations."* [22]. A BP consists of tasks connected together with respect to a particular process model that BP engineers define at design-time. At run-time, the process's tasks are assigned to persons/machines for manual/automated execution. Different works suggest fragmenting BPs and deploying them over the clouds to tap into their elasticity and pay-as-you-go benefits [23]. However, deploying fragmented BPs over the cloud [14] could fail when constraints like temporal are not satisfied resulting into financial penalties, for example.

In line with the works on cloud-driven BP fragmentation, we presented in [5] an approach to formally specify and verify cloud resources allocation to BPs using Timed Petri-Net (TPN). Our objective was to ensure that this allocation does not violate any temporal constraints on BPs. We also presented how BP correctness is formally verified with respect to such constraints. In this paper we extend this approach by fragmenting and deploying free-of-violations time-constrained BPs in a mono-cloud and multi-edge context. We resort to cloud-edge collaboration by verifying at both design-time and run-time where data should be placed (cloud, edge, or either) and where tasks should run as well (cloud, edge, or either). Satisfying temporal constraints in the context of cloud is thoroughly discussed in the literature [4, 12, 13, 19]. However, this remains unexplored in the context of cloud/edge, which constitutes one of our main contributions in this paper.

The remainder of this paper is organized as follows. Section 2 introduces a motivating example. Section 3 briefly presents the approach for verification of cloud- and edge-based resource allocation at both design-time and run-time. We detail the design-time stage in Section 4 and the run-time stage in Section 5. Section 6 gives implementation details. Section 7 discusses related work. Finally, concluding remarks and future work are drawn and presented, respectively, in Section 8.

2. Motivating example

Timely responses to customers' requests are a key competitive advantage in any economy. Many organizations tap into *lead time* to sustain this advantage despite the existence of many factors that they cannot, sometimes, control like unannounced strikes and bad weather. Consequences of late delivery are multiple ranging from financial penalties to angry customers and market-share shrinkage. Amazon.com perfectly illustrates how this giant online-retailer achieves its targets by offering competitive prices despite limited lead times. Amazon.com embraces many ICT gadgets like drones in conjunction with customer care best-practices like return policies. On December 7th, 2016, Amazon.com announced its first drone-based air delivery in Cambridge, UK with a shipment lasting 13 minutes from purchase to drop-off[4]. This would not have happened if Amazon.com

[4] www.engadget.com/2016/12/14/amazon-completes-its-first-drone-powered-delivery. www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

does not operate warehouses in different parts of the world, Cambridge in this case. The objective is to be “close” to customers to avoid potential delivery delays. Collecting data on shipping BPs for the sake of tracking could also benefit from the concept of “closeness” that edge computing promotes. Indeed, sending these data to remote cloud servers for processing could limit Amazon.com’s promptness. By the time the data arrives to these servers, it could become obsolete, be subject to unauthorized collection, and have consumed unnecessary bandwidth. This becomes acute in the case of drone delivery when instant response to unforeseen events (e.g., wind speed) is a must. In the rest of this paper, we illustrate how a delivery BP could benefit from cloud/edge collaboration. To this end, a set of what-if analysis will be carried out to decide which parts of the BP should reside in the cloud, which parts of the BP should reside in the edge, and which parts of the BP should reside in either. This what-if analysis along with cloud/edge collaboration will be illustrated in the rest of this paper using drone-based delivery BP whose tasks are listed in Table 1.

Table 1. Tasks defining drone-based delivery BP

Task	Description
t_1	collect and verify delivery instructions
t_2	pick items from robot-smart warehouse
t_3	define drone position
t_4	process drone position and other details
t_5	update logbook of flying/delivering drones
t_6	analyse received information
t_7	send instructions to drone to avoid accidents or violations
t_8	update drone position
t_9	notify customer about item arrival

3. Approach in a nutshell

This section introduces the main concepts and definitions related to TPNs. Afterwards, it presents the foundations of the proposed approach.

3.1. Time Petri-Nets

A PN is formed upon a mathematical theory that uses automated tools to offer an accurate modeling and analysis of systems’ behaviors [3]. Initially, PNs were a formal language without any reference to time or probability. However, for many critical applications, time is a must-have and designers should consider it when analyzing the correct behavior and performance of these applications. TPNs integrate clocks and temporal constraints into transitions to help describe and analyze properly time-dependent systems. TPNs associate a firing time interval [a,b] with each transition, where a and b are rational numbers such that $0 \leq a \leq b$ and $a \neq \infty$. Times a and b for t are relative to the moment at which t was enabled; a and b are referred to as earliest-firing-time and latest-firing-time of t, respectively. Formally, a TPN is a tuple Y = (P, Tr, Pre, Post, M_0, IS) where (P, Tr, Pre, Post, M_0) is a PN and IS: Tr→ $Q^* * (Q^* \cup \{\infty\})$ is a static interval function that associates each Tr with a time interval IS (Tr)= [min,max] so, that, Q^* is the set of positive rational numbers. Readers are referred to [3] for more details about TPNs.

3.2. Foundations

Our approach puts forward recommendations about the "ideal" placement of a BP's tasks and data in a mono-cloud, multi-edge context. Should the recommendations violate any time constraint at design- and/or run-time, then the violations should have a limited impact on completing the BP. By limited we mean an acceptable overtime with respect to some thresholds that BP engineers could set, e.g., 10 extra minutes are still acceptable. Our approach spans over BP design-time (Fig. 1) and run-time (Fig. 2) having each a set of dedicated stages.

At design-time, 4 stages namely *specification*, *placement*, *transformation*, and *verification*, take place.

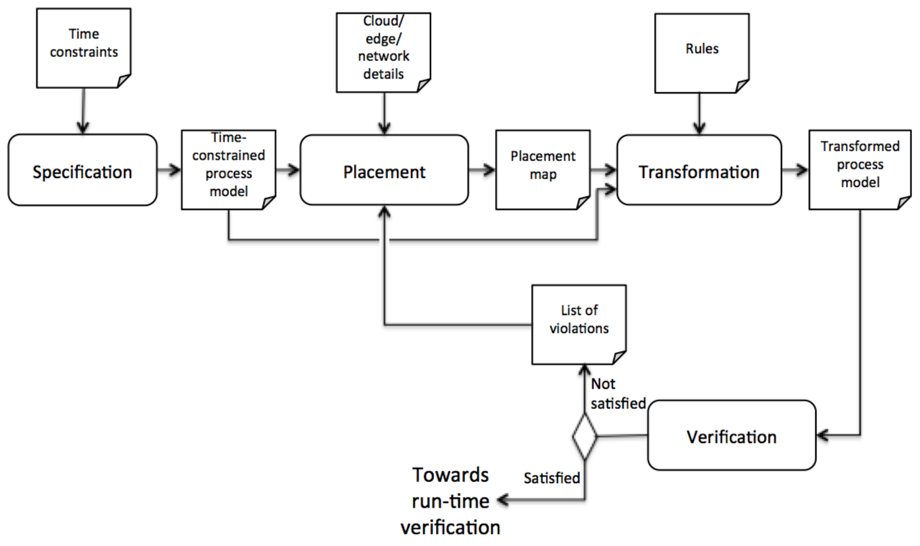


Fig. 1. Stages of the approach at design-time

The *specification* stage consists of modeling the BP enriched with time constraints on tasks and data. The *placement* stage takes as input the time-constrained process model and available cloud and edge devices to assign this BP's tasks and data to these devices. This assignment is reported into what we call a placement map. The *transformation stage* uses some in-house rules to convert the placement map into *a transformed process model* that is, in fact, a TPN. Our rules address Business Process Model and Notation (BPMN) constructs, temporal constraints, and time related to transferring data from cloud to edge and *vice-versa*. Finally, the *verification* stage checks if the *TPN process model* contains any time violations that needs to be fixed by going back to the *placement* stage. Otherwise, we proceed with the run-time verification.

At run-time, 3 stages namely *execution*, *ongoing verification*, and *ongoing placement* take place. A BP execution engine will run BP instances. A *log* is also used during run-time to capture the BP's instance execution progress and outcomes.

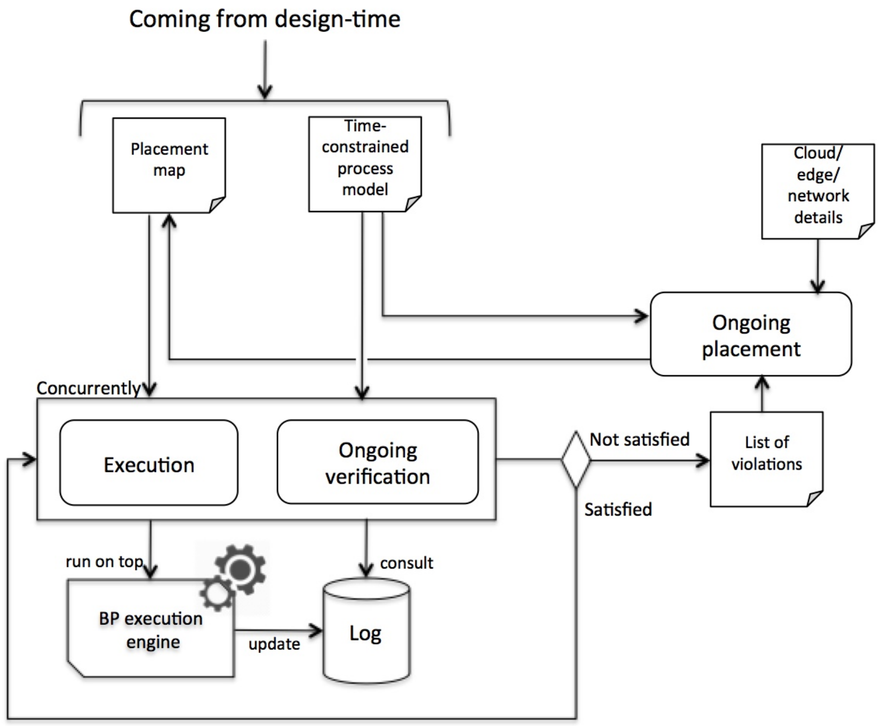


Fig. 2. Stages of the approach at run-time

The *execution* stage instantiates the BP's process model so, that, process instances are run over an execution engine. Concurrently to the *execution* stage, the under-execution BP instances are subject to verification to ensure that no time-constraints are violated. This *ongoing verification* stage could produce a *list of violations* so, that, corrective actions are taken to avoid additional violations that could impact the rest of tasks and/or data, and hence, more penalties. The corrective actions are identified thanks to the *ongoing placement* stage that will put forward additional recommendations about potential changes related to where future tasks and/or data should be re-placed. The implementation of these recommendations will be again reflected on the placement map.
]

4. Design-time specification and verification

Since the definition of a BP's temporal constraints over tasks and data during the *specification* stage may lead to deadlocks or unmet deadlines, designers could resort to formal languages to ensure this definition correctness. In this section, we detail further the design-time *transformation* and *verification* stages.

4.1. Specification stage

The engineer designs the BP's model using any modeling language, although we recommend BPMN. Then, he defines the time constraints on this BP's tasks [7, 8] and data.

1. Temporal constraints on tasks. First, we consider *intra-task* temporal constraint of type execution *duration*. Let s(t) (resp., e(t)) be the starting (resp., the ending) time of a task t. Let d be a relative time value representing the duration of this task. *Duration* constraint is defined as per Equation 1:

$$\textit{Duration}(\text{t,d}) \overset{\text{def}}{=} e(t) - s(t) \leq d \tag{1}$$

   Afterwards, we consider *inter-task* temporal constraints of type execution *dependency* that could be:
   - Start-to-Finish (SF): t_j can not finish until t_i has started within a given time interval.
   - Start-to-Start (SS): t_j can not begin before t_i starts within a time interval.
   - Finish-to-Start (FS): t_j can not begin before t_i ends within a time interval. As per Equation 2, t_j should start its execution no later than $MaxD$ time units and no earlier than $MinD$ time units after t_i ends.

$$TD(FS, t_i, t_j, MinD, MaxD) \overset{\text{def}}{=} MinD \leq s(t_j) - e(t_i) \leq MaxD \tag{2}$$

   - Finish-to-Finish (FF): t_j can not finish until t_i has finished within a time interval.
2. Temporal constraints on data. We consider freshness to define the allowable time for a data to remain valid (adapted from [16]) when it is exchanged between tasks that could be running in separate hosts (e.g., any data received 2 seconds from its expected arrival time is not valid). Let t_i be the task producing data $data_{ij}$ that task t_j will consume. Formally, data freshness is defined as per Equation 3:

$$\begin{aligned} FreshData(data_{ij}, t_i, t_j, MaxValue) \overset{\text{def}}{=} &\textit{produce}(t_i, data_{ij}) \\ \&\textit{consume}(t_j, data_{ij}) \& s(t_j) - e(t_i) &\leq MaxValue \end{aligned} \tag{3}$$

Let's recall the case study. The engineer can propose the temporal constraints on BP's tasks and data as per Table 2. For example, FS temporal dependency is specified between t_4 and t_8. In addition, the data that t_4 produces needs to be consumed by t_5 in less than 5 seconds.

Table 2. Case study's temporal constraints on tasks and data

task	duration	temporal dependency	data freshness
t_1	10 milliseconds		
t_2	10 milliseconds	TD(FS,t_1,t_2,1 millisecond,5 milliseconds)	
t_3	12 minutes		
t_4	10 milliseconds		
t_5	10 milliseconds		FreshData(t_4,t_5,5 seconds)
t_6	10 milliseconds		
t_7	10 milliseconds		FreshData(t_6,t_7,5 seconds)
t_8	10 milliseconds	TD(FS,t_4,t_8,1 millisecond,8 milliseconds)	FreshData(t_7,t_8,5 seconds)
t_9	10 milliseconds		

4.2. Placement stage

The engineer proceeds with a first assignment of the BP's tasks and data to cloud/edges. This assignment results into a *placement map* that considers how critical the tasks are (i.e., must execute), how dependent the tasks are, what time constraints are imposed on tasks and data (e.g., duration and freshness), and any other time-related details (e.g., data transfer between hosts). How good this first assignment will be checked out during the *verification* stage (e.g., no time constraints' violation). However it is worth noting that the *placement* stage could be triggered again if the *verification* stage reports any temporal violation. These violations are manually analyzed by BP engineers to identify their causes. Consequently, the engineer can come along with some corrective actions that would address these violations like reassigning some tasks/data from cloud to edge and *vice versa*. It is highly recommended to address any violation prior to executing the BP.

Table 3 presents an initial placement that the BP engineer could come up with. For example, "send instructions to drone to avoid accidents or violations" task is deployed on the edge device to be each time near to the drone while flying since the decision of changing position of drone should be made as quickly as possible. Indeed, The use of edge nodes is to reduce the overall response time and traffic to distant clouds. Contrarily, sending instructions to edge nodes will be forwarded to " update logbook of flying/delivering drones" task, which is deployed on the cloud, where high volumes of data can be processed.

Table 3. Initial placement of drone-based delivery BP's tasks

task	placement
t_1, t_2, t_5, t_6, t_9	cloud
t_4, t_7	edge

4.3. Transformation stage

This stage relies on a set of *rules* defined in compliance with model-driven engineering principles. The objective is to convert a time-constrained BPMN process into a TPN model.

It all begins by transforming BPMN basic elements like start/end events, tasks, and gateways into TPN. Readers are referred to [6] for a complete description of the transformation rules. We focus on rules related to BP task placement. For instance, a task with an execution duration will be transformed into 1 place and 2 transitions labeled with clocks depending on the task's duration (Fig. 3 (a)). The duration is set by the engineer depending on whether the task will run on either the cloud or the edge as per the *placement map*. Fig. 3 also includes the transformation of some temporal dependencies between tasks such as SS, FS, and FF.
Let us now focus on the transformation rules for data placement. For a given data $data_i$, it can be produced by a task t_i that runs on a certain host (h_c to refer to cloud) and will be consumed by another task t_j that runs on a different host (h_y to refer to edge). In this case, $data_i$ should be transferred from one host to another and hence, *latency* time ($l^{d_i}_{h_x:h_y}$) needs to be considered. Data latency is transformed into 2 places and 1 transition as per Fig. 3 (b).

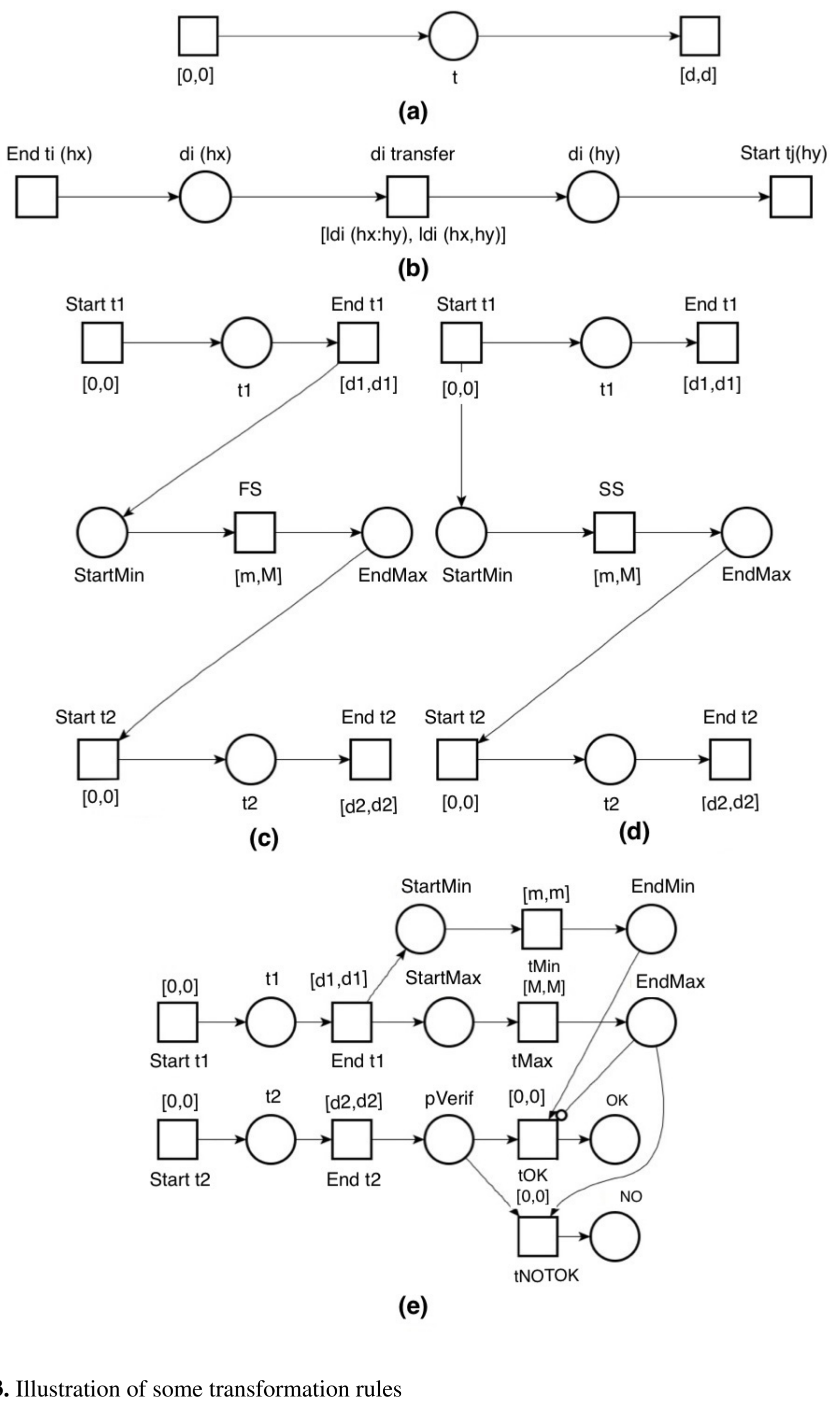


Fig. 3. Illustration of some transformation rules

4.4. Verification stage

Depending on a BP complexity, software testing and/or manual checks could turn out insufficient and hence, time constraints violations could still arise. To address this concern, we propose model checking as an accurate and exhaustive verification alternative [2]. The BP engineer checks the correctness of the time-constrained, TPN-based process model using a model checker like TIme petri Net Analyzer (TINA) [3].

If the check reports any temporal violation, a *list of violations* (could be based on counter-examples) is reported and the designer is referred back to the *placement* stage. At this stage, a threshold could be put to limit the number of times the designer has to initiate this stage. Otherwise, the designer proceeds with executing the BP. Violations at this stage could refer to deadlocks due to temporal inconsistencies (e.g., a task minimum duration exceeding the maximum delay of initiating a dependent task) and missed deadlines (e.g., process ending in 5h but the designer has set 4h as a maximum).

Concretely, we formally verify a TPN-based process model with respect to the following properties: deadlock freshness, process deadlines, and data freshness. These properties are written in State/Event Linear time Temporal Logic (S/E LTL [17]).

– $\Diamond$ (- dead) : to check that a process is free of deadlocks.
– $\Diamond$ (- notdeadline_process) : to verify if a deadline has been met. This means that notdeadline_process place (associated with an observer for the deadline property) is false throughout the whole path leading to this place.
– $\Diamond$ (- notfresh_data) : to verify if the freshness time f^{data_i} related to $data_i$ has been met. This means that notfresh_data place (associated with an observer for the *freshness* property) is false throughout the whole path leading to this place.

The verification of the latter properties on the generated TPN of the case study reports that the process is deadlock free, and meets the deadline (18 minutes), and all data respect their freshness constraints.

Despite the virtues of design-time model-checking that could guarantee certain free-of-time violations, many run-time events and actions could happen triggering such violations. For instance, expected duration times could suddenly change due to power outage and hence, raising questions about the validity of design-time model-checking. To this end, we propose run-time verification to monitor process execution. In other words, verification at design-time is "preventive" rather than "curative", which backs our run-time verification.

5. Run-time verification

During execution, current time values like duration and freshness may change resulting into gaps with the estimated values and thus, can violate time constraints (e.g., a high latency can make data obsolete and a late data arrival can delay a task execution). Thus, BP instances need to be continuously monitored to ensure the satisfaction of their time constraints at run-time. In this section, we detail further run-time's different stages (Fig. 2).

5.1. Execution stage

A BP instance runs on top of an execution engine that assigns tasks to persons/machines, transfers data between tasks, stores data, etc. Both the *time-constrained process model* and the *placement map* constitute inputs to the *execution* stage that continuously updates the *log repository* that contains details about instances execution like instance id, exchanged data, and execution outcome (success or failure).

5.2. Ongoing verification stage

Because of the dynamic nature of environments in which BP instances are executed, we adopt thresholds that would give engineers some leeway (i.e., extra time) prior to raising the violation flag. In project management [15], this leeway is known as slack time. We tap into our previous work on service execution [11] to define thresholds with respect to a constraint satisfaction model that captures both task duration and data transfer that impacts data freshness (Constraints (4) to (13)). Our objective is to recommend a maximum threshold for task duration while guaranteeing data freshness.

$$\text{maximize } Duration(t_j) \tag{4}$$

$$Agg_{t_i \in \mathcal{T}}(Duration(t_i)) \leq deadline, \forall t_i \in \mathcal{T} \tag{5}$$

$$e(t_i) \leq s(t_k), \forall t_k \in \mathcal{T}, t_i \in \mathcal{P}d(t_k) \tag{6}$$

$$Duration(t_i) = EstimatedDuration(t_i), \forall t_i \in \mathcal{T}, i \neq j \tag{7}$$

$$s(t_i) + Duration(t_i) = e(t_i), \forall t_i \in \mathcal{T} \tag{8}$$

$$e(t_i) + MinD \leq s(t_k),\ \forall\, TD(FS, t_i, t_k, MinD, MaxD) \in \mathcal{TD} \tag{9}$$

$$s(t_k) \leq e(t_i) + MaxD,\ \forall\, TD(FS, t_i, t_k, MinD, MaxD) \in \mathcal{TD} \tag{10}$$

$$e(t_i) + Transfer(d_i) \leq s(t_k),\ \forall\, DD(d_i, t_i, t_k, h_i^s, h_k^s) \in \mathcal{DD} \tag{11}$$

$$Duration(t_j) \in [EstimatedDuration(t_j), deadline] \tag{12}$$

$$st_i, et_i \in [0, deadline], \forall t_i \in \mathcal{T} \tag{13}$$

The maximum threshold of each task t_j is equal to its maximum allowed duration value (i.e., $Duration(t_j)$). Constraint (5) guarantees that the global duration constraint is satisfied. The aggregation function *Agg* depends on the distinguish characteristics of quality attributes (i.e., additive, average, multiplicative, and max-Operator) and the structure of the BP (i.e., structural patterns involved such as sequence, parallel, choice, and loop). Here, the duration is considered as max-operator quality attribute. Hence, the aggregation function is as follows:

- $Agg_{t_i \in \mathcal{T}}(Duration(t_i)) = \sum_{i=1}^{n} Duration(t_i)$ for sequence patterns where n is the number of tasks in the sequence pattern.
- $Agg_{t_i \in \mathcal{T}}(Duration(t_i)) = max_{i=1}^{n}\{Duration(t_i)\}$ for parallel patterns where n is the number of tasks in the parallel pattern.
- $Agg_{t_i \in \mathcal{T}}(Duration(t_i)) = Duration(t_k)$ for choice patterns where t_k is the selected task in the choice pattern.

– $Agg_{t_i \in \mathcal{T}}(Duration(t_i)) = \alpha_i Duration(t_i)$ for loop patterns where α_i is the number of loops of the task t_i.

Constraint (6) deals with dependencies between tasks where $\mathcal{P}d(t_k)$ denotes the set of immediate predecessors of the task t_k and $s(t_i)$ and $e(t_i)$ denote the start time and end time of the task t_i, respectively. The duration of each task t_i is equal to the estimated duration specified at design time (Constraint (7)). Moreover, the end time of each task t_i is equal to the sum of its start time and its duration (Constraint (8)). To deal with temporal dependencies between tasks, we use Constraints (9) and (10) where $\mathcal{TD}$ is the set of temporal dependency. For simplicity, we consider only finish-to-start dependencies. Constraint (11) guarantees data freshness where $\mathcal{DD}$ is the set of data dependencies between tasks. $DD(data_i, t_i, t_k, h_i^s, h_k^s)$ denotes the data dependency between the task that produces the data $data_i$ (i.e., task t_i) and the task that consumes the data $data_i$ (i.e., task t_k) when t_i is executed in the host h_i^s and t_k is executed in the host h_k^s. The time required to the transfer of the data $data_i$ from t_i to t_k is denoted by $Transfer(data_i)$. The domain of the maximum duration threshold and the start and the end time are presented in Constraints (12) and (13), respectively.

The maximum threshold of each task t_i is denoted by T_i^M. During execution, if one of the maximum thresholds is exceeded, the global duration is violated and thus, corrective actions should be taken which will be discussed in the ongoing placement stage. We note that maximum thresholds have to be recomputed after each violation.

In conjunction with the maximum thresholds, we identify a set of intermediary thresholds for all tasks. They are used to trigger placement actions prior to a global constraint violation. Each time a deviation exceeds an intermediary threshold, the placement of tasks is adjusted so, that, possible violations in the remaining non-executed tasks can be either reduced or prevented. The aim is to avoid delaying the placement until a violation of a global constraint occurs on the one hand, and, on the other hand avoid triggering placement actions each time a violation is observed, which can decrease the efficiency of the proposed approach. The intermediary threshold of each task t_i is denoted by T_i^I that is the average between the estimated duration value (before the execution) and the maximum threshold of the same task.

5.3. Ongoing placement

To ensure a continuous execution of the different BP instances while guaranteeing the satisfaction of all constraints, the BP instances need to continuously react to varying conditions during execution. We present hereafter the ongoing placement of tasks and data each time a deviation of an intermediary threshold or a violation occurs. To enhance the efficiency of the placement, we identify a set of alternative hosts for each task. Thus, a local placement can be easily applied to change the placement of tasks/data and guarantee the satisfaction of the different constraints. In case of a deviation/violation during execution, we propose to change the host of one or more tasks using the alternative hosts. We note that alternative hosts are updated and re-identified during execution each time a change occurs in the BP instances.

Alternative hosts Prior to execution, we define a set of alternative hosts for each task $t_i \in \mathcal{T}$ denoted by H_{alti}. Alternative hosts should satisfy the maximum thresholds of their corresponding tasks. Hence, an alternative host $h_j \in H_{alti}$ enables to execute task t_i to start and end its execution without exceeding the maximum temporal thresholds (i.e., $Duration(t_i) \leq T_i^M$) and guarantee the freshness of data. In fact, since we check the satisfaction of the transfer time when computing maximum thresholds (Constraint (11) in the model from Constraints (4) to (13)), all hosts that satisfy the maximum thresholds will guarantee the satisfaction of the task duration and data freshness at the same time. A host h_j that satisfies the maximum thresholds of its corresponding task is denoted by (h_j sat T_i^M). Contrarily, $\neg(h_j$ sat $T_i^M)$ denotes that the maximum threshold is not satisfied by the host h_j. In this latter case, the host will not be considered in the set of alternative hosts of its corresponding task. We note that the set of alternative hosts is updated each time a deviation or a violation occurs to take into account the new values of the duration of the already executed tasks and the new values of the maximum thresholds. We rank the set of alternative hosts based on the duration of the execution of the tasks in each host. Hence, the host that guarantees the minimum execution duration will be ranked first and so on. We denote by $Duration(t_i)^{h_j}$ the execution duration of the task t_i when it is executed in host h_j.

Changing hosts We denote by $PM^* = \{h_1^s, ..., h_i^s, ..., h_n^s\}$ the placement map with h_i^s is the selected host for the task t_i. By ongoing placement denoted by PM^*_{new}, we refer to changes in the placement map.

Algorithm 1 handles changes in the ongoing placement map. If the execution time of task t_i, while being executed in host h_i deviates but does not exceed the intermediary threshold (T_i^I), then this will not affect affect the placement map (lines 4 to 6). If the deviation is between the intermediary and maximum thresholds (line 7), then we proceed with first, the maximum thresholds and the set of alternative hosts H_{alti} are updated for each non-executed task considering the values of the already executed tasks (lines 8 and 9) where T_{ne} denotes the set of non-executed tasks. We note that when updating the thresholds, the execution duration of the already executed tasks are considered in the constraint satisfaction model from (4) to (13) (Section 5.2). Moreover, the set of alternative hosts is updated by identifying the new alternative hosts based on the new values of maximum thresholds. Then, if the first alternative host guarantees a better execution duration than the initial selected host, it will be considered in the ongoing placement map (lines 10 to 12) where h_i^1 denotes the best host for the task t_i when considering the already executed tasks. The aim of this step is to avoid the accumulation of deviations during execution in order to prevent possible violations. If a violation exceeds the maximum threshold (line 15), then, the ongoing execution is no more satisfactory. In this case, we update the maximum thresholds and alternative hosts for the non executed tasks (line 17). If there is at least one host in the set of alternative hosts for a non-executed task, the selected host of this task will be substituted by the first alternative host (lines 18 to 23).

If the ongoing placement is modified, all thresholds and alternative hosts for all non-executed tasks will be updated (lines 26 to 30).

Algorithm 1 Ongoing placement

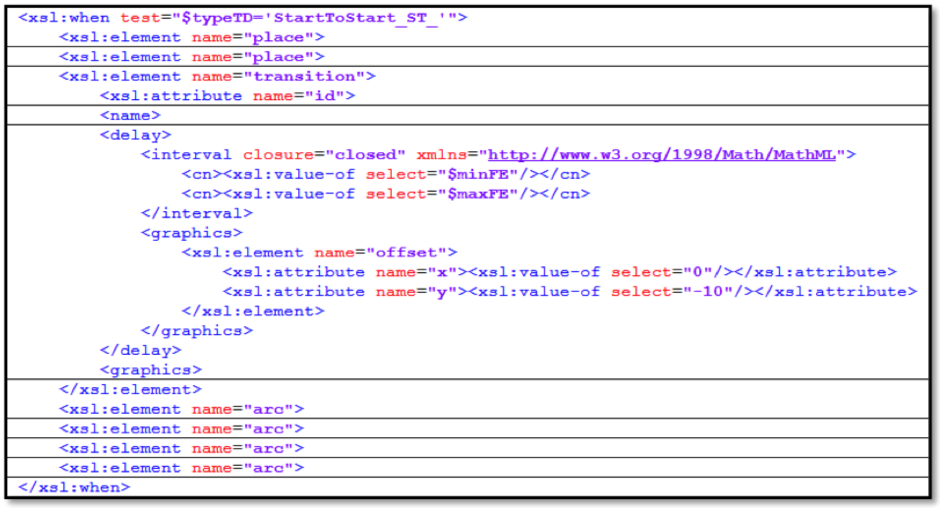
```
1: Input: Monitoring results of task $t_i$, the placement map $PM^*$
2: Output: The new placement map $PM^*_{new}$
3: $PM^*_{new} = \emptyset$
4: if ($h_i^s$ sat $T_i^I$) then
5:     $PM^*_{new} = PM^*$
6: end if
7: if $\neg$($h_i^s$ sat $T_i^I$) and ($h_i^s$ sat $T_i^M$) then
8:     for each $t_i \in T_{ne}$ do
9:         update($T_i^M, H_{alti}$)
10:        if $Duration(t_i)^{h_i^1} < Duration(t_i)^{h_i^s}$ then
11:            $PM^*_{new} = PM^* \setminus \{h_i^s\} \cup \{h_i^1\}$
12:        end if
13:    end for
14: end if
15: if $\neg$($h_i^s$ sat $T_i^M$) then
16:    while $PM^*_{new} = \emptyset$ and $t_i \in T_{ne}$ do
17:        update($T_i^M, H_{alti}$)
18:        if $H_{alti} \neq \emptyset$ then
19:            $PM^*_{new} = PM^* \setminus \{h_i^s\} \cup \{h_i^1\}$
20:            break;
21:        else
22:            move to $T_{i+1}$
23:        end if
24:    end while
25: end if
26: if $PM^*_{new} \neq \emptyset$ then
27:    for each $t_i \in T_{ne}$ do
28:        update($T_i^I, T_i^M, H_{alti}$)
29:    end for
30: end if
```

6. Implementation

This section discusses the implementation work that was carried out in terms of experiments and performance evaluation. In compliance with how our approach is designed, we discuss the implementation at design-time and then run-time.

6.1. Design-time implementation

We extended the work we presented in [13] that resulted into the development of an Eclipse plug-in. Using this plug-in, a designer represents a BP's 2 things: needs of resources (cloud and/or edge resources) and time-constrained activities. Next, a source model is generated as an XML document. In our work, we focused on implementing rules that transform BPs into TPN. This is done using an XSLT file containing the transformation rules. Fig. 4 exhibits an XSLT excerpt that transforms a SS temporal dependency into 2 places and 1 transition with a delay of minFE and maxFE.



```
<xsl:when test="$typeTD='StartToStart_ST_'">
    <xsl:element name="place">
    <xsl:element name="place">
    <xsl:element name="transition">
        <xsl:attribute name="id">
        <name>
        <delay>
            <interval closure="closed" xmlns="http://www.w3.org/1998/Math/MathML">
                <cn><xsl:value-of select="$minFE"/></cn>
                <cn><xsl:value-of select="$maxFE"/></cn>
            </interval>
            <graphics>
                <xsl:element name="offset">
                    <xsl:attribute name="x"><xsl:value-of select="0"/></xsl:attribute>
                    <xsl:attribute name="y"><xsl:value-of select="-10"/></xsl:attribute>
                </xsl:element>
            </graphics>
        </delay>
        <graphics>
    </xsl:element>
    <xsl:element name="arc">
    <xsl:element name="arc">
    <xsl:element name="arc">
    <xsl:element name="arc">
</xsl:when>
```

Fig. 4. Transformation rule of a SS temporal dependency as XSLT

The result of the transformation is an XML document that describes the generated TPN. An example of this TPN based on the drone delivery is given in Fig. 5. *data1Edge* and *data1Cloud* places and *Tlatency1* transition labeled by the time interval [latencyEC,latencyEC] specify data transfer time between t_4 (running on edge) and t_5 (running on cloud). We consider "latencyEC" equals to 200 milliseconds as latency edge to cloud. *StartMax2* and *EndMax2* places and *FS2* transition labeled by the time interval [m2,M2] with m2= 1 millisecond and M2= 8 milliseconds specify a temporal dependency constraint between t_4 and t_8.

Finally, we formally verify the matching between the activities, temporal constraints, and resource temporal constraints. The generated TPN are the inputs for the TINA model checker.

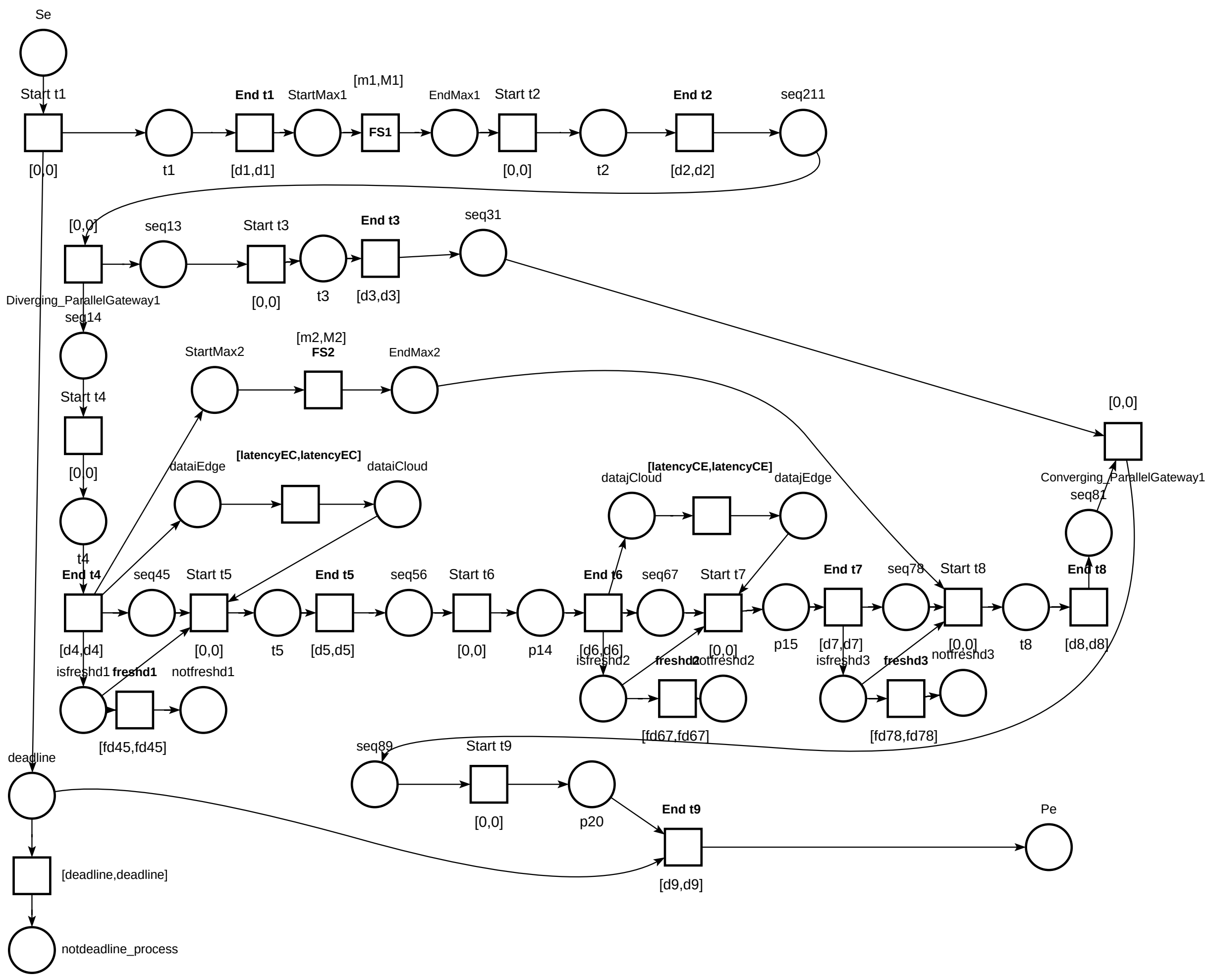


Fig. 5. Generated TPN of the drone delivery BP

6.2. Run-time implementation

We investigated how our approach behaves at run-time. First, we evaluate the success rate (I suggest to remove this. This is not the definition of success rate. i.e., converging “quickly” to optimal solution) and computation time. Thus, constraints (4) to (13) and algorithm 1 were used to test a BP of 9 tasks generated randomly. As candidate hosts, we used a mono-cloud with multiple VMs and 10 edges respectively. Constraint satisfaction models are implemented using the constraint solver Choco[5].

First, the success rate is compared to First In First Out (FIFO), in which, the first come host is first assigned to task without taking into account the host that has the best duration and the replacement of hosts is delayed until a global violation occurs which can affect the

[5] http://www.emn.fr/z-info/choco-solver/

execution of tasks contrarily to our approach which allows enhancing the selected hosts during execution as soon as a deviation occurs.

Fig. 6 depicts the success rate in response to the number of deviations in process tasks which are generated randomly at run-time. All deviations are assumed to be less than the maximum thresholds (see subsection 5.2). The positions of deviations are generated randomly. Experimental results show that our approach has a higher success rate in comparison to FIFO approach. Indeed, it reacts to changes as soon as they occur which increases the likelihood to find a solution. In contrast to FIFO, it might be the case where no solution is found after a violation which can be caused by multiple deviations.

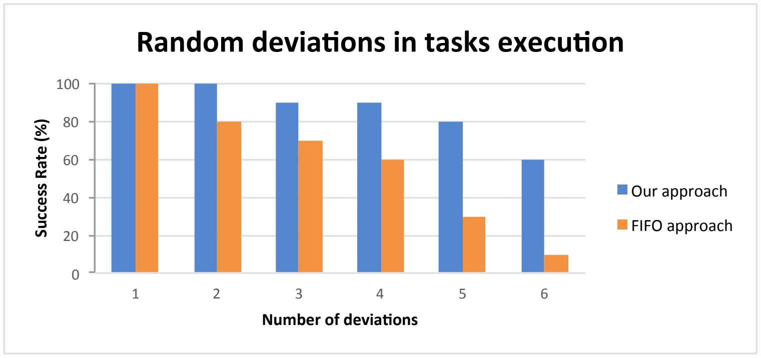


Fig. 6. Success rate versus number of deviations

Second, we calculate the computation time of our approach. It takes between 7 and 200 milliseconds to find a solution. These time values are taken while considering random task violations exceeding the maximum thresholds and thus, hosts changing is mandatory to guarantee the satisfaction of all process constraints. Indeed, solutions can be found by changing hosts using the alternative hosts (see subsection 5.3). In addition, new placement actions are taken as soon as deviation occurs in parallel to the execution and does not cause the interruption of the execution. Results show that the computation time of our approach is “negligible” compared to the deadline of the expected process. Results also show that the computation time increases proportionally to the number of deviations in the process.

7. Related work

Our related work consists of 2 parts. The first part is about BP formal specification. The second part is about BP allocation into clouds. Many works in the literature address the issue of defining BP formal specification. First, Dijkman et al. in [9] propose a formal BPMN semantics defined in terms of a transformation to standard PN. The transformation has been implemented as a tool that generates Petri Net Markup Language (PNML) code. But, the authors do not consider any temporal dimension in their analysis. Rachdi et al. [19], propose an approach that takes into account time concepts in BPMN processes. They present a formal semantics of BPMN defined in terms of transformation to TPN but without taking into consideration of temporal constraints as in our work nor the notion of resources. Cheikhrouhou et al.[7, 8] address the problem of formal specification and verification of temporal constraints of activities using timed automata. But, resources were not considered. Hachicha et al [12] extend of the BPMN meta-model to optimally manage cloud resources. They formalize the resources consumed using a shared knowledge base. Therefore, the authors propose a semantic framework for BPs enriched by cloud resources. However, the temporal perspective for BPs is out of reach.

Several works have addressed the specification and formal verification of cloud resources in BPMN. Boubaker et al.[4] validate the consistency of the allocation of cloud resources using Event- B. The latter is used to formally specify cloud resource allocation policies in business process models and to verify its accuracy based on user requirements and resource properties. However, in this work, BPs are not enriched by time constraints. Ben Halima et al. [13] formally specify temporal constraints on pricing strategies for cloud resources, especially virtual machines, and on BPMN activities. This specification is translated into timed automata to formally verify the correspondence between the time constraints of the business process and the cloud resources. But, this work does not deal with constraints on process data nor support automatic BPMN mapping to timed automata, which can lead to errors during the transformation. Several searches extend BPMN with time constraints and cloud resource perspectives and use formal verification. Watahiki et al. [21] extend BPMN to handle time constraints. They also provide an automatic mapping of extended BPMN to timed automata. This approach aims to verify certain characteristics, such as deadlock. However, the scope of this article is limited to a small subset of BPMN elements. In addition, the extension proposed in this work gives specific temporal constraints to a single task of the business process model and does not take into account time constraints related to a set of activities such as temporal dependency. There is previous research that aims to check whether the selected cloud resource meets the time constraints of business processes. Du et al.[10] propose to dynamically verify the temporal constraints of multiple simultaneous business processes with resources. However, the work does neither deal with data flow and their temporal constraints, nor with edge resources. While almost works in the litterature focus only on control flow verification, process data flow modeling is of similar importance. In [1], the approach generates a PN process model that captures the control flow along with data aspects of BPMN process models. The approach detects data-flow errors in BPMN 2.0 process models, such as missing or unused data and possible deadlocks in the PN model. However, the approach does not deal with temporal constraints on process data. The approach in [18] focuses on the resource allocation problem in fog computing based on Priced Timed Petri nets (PTPN). Provided with a group of pre-allocated resources, the designer can choose the satisfying resources autonomously while considering both the price and the cost to execute a process's tasks as the credibility evaluation of both users and fog resources. From one hand, the constructed PTPN models of process tasks does not deal with temporal constraints of the process such as deadline nor with cloud resources and the delay caused by data transfer from one host to another. Furthermore, the PTPNs were used as a formal background for a proposed algorithm that predicts task completion time. Thus, no formal verification is proposed and simulation results are presented. To the best of our knowledge, there is no research attempts to verify process models while addressing both cloud and edge resource allocation, data flow aspects, and their temporal constraints. Such verification is scarce at both design-time and run-time.

8. Conclusion

This paper presented an approach to specify, verify, and deploy BPs in a mono-cloud, multi-edge context. These BPs are bound to time constraints whose satisfaction requires placing their tasks and data in the appropriate hosts, whether cloud, edge, or either. This placement is continuous because of the dynamic environment in which BPs are expected to execute. Indeed, communication networks could become jammed and some computation resources could become unavailable. Either reason could lead to delays in executing tasks and/or transferring data. Delays raise time violations, which themselves mean penalties of all types, financial, market share loss, etc. Our specification, verification, and deployment approach happens at both BP design-time and BP run-time involving different stages such as specification, ongoing placement, verification, and execution. One of the run-time stages, ongoing placement, included a set of thresholds that give BP engineers some leeway (i.e., extra time) prior to raising any violation flag. In term of future work, we would like to extend the proposed approach to deal with several simultaneous changes in a BP's tasks and data placement and propose strategies to handle potential conflicts between corrective actions. Furthermore, we aim to further compare our approach to other approaches that suggest backup solutions.

References

1. Ahmed Awad, Gero Decker, and Niels Lohmann. Diagnosing and repairing data anomalies in process models. In Stefanie Rinderle-Ma, Shazia Sadiq, and Frank Leymann, editors, *Business Process Management Workshops*, pages 5–16. Springer Berlin Heidelberg, 2010.
2. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.
3. Bernard Berthomieu and François Vernadat. Time petri nets analysis with TINA. In *Proceedings of the Third International Conference on the Quantitative Evaluation of Systems (QEST*.
4. Souha Boubaker, Walid Gaaloul, Mohamed Graiet, and Nejib Ben Hadj-Alouane. Event-b based approach for verifying cloud resource allocation in business process. In *Proceedings of the 2015 IEEE International Conference on Services Computing, SCC*, pages 538–545, 2015.
5. Saoussen Cheikhrouhou, Nesrine Chabouh, Slim Kallel, and Zakaria Maamar. Formal specification and verification of cloud resource allocation using timed petri-nets. In *Proceedings of the New Trends in Model and Data Engineering - MEDI 2018 International Workshops, DETECT, MEDI4SG, IWCFS, REMEDY, Marrakesh, Morocco, October 24-26, 2018*, pages 40–49, 2018.
6. Saoussen Cheikhrouhou, Nesrine Chabouh, Slim Kallel, and Zakaria Maamar. Transformation of timed BPMN busines processes and cloud resources into timed Petri-Nets. Technical report, http://www.redcad.tn/projects/bpmn2tpn/technicalreport-0618.pdf, 2018.
7. Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. Toward a time-centric modeling of business processes in BPMN 2.0. In *The 15th International Conference on Information Integration and Web-based Applications & Services, IIWAS*, page 154, 2013.
8. Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015.

9. Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Formal semantics and analysis of bpmn process models using petri nets. *Queensland University of Technology, Tech. Rep*, 2007.
10. YanHua Du, PengCheng Xiong, YuShun Fan, and Xitong Li. Dynamic checking and solution to temporal violations in concurrent workflow processes. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(6):1166–1181, 2011.
11. I. Guidara, I. Al Jaouhari, and N. Guermouche. Dynamic selection for service composition based on temporal and qos constraints. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 267–274, June 2016.
12. Emna Hachicha and Walid Gaaloul. Towards resource-aware business process development in the cloud. In *Proceedings of the 29th IEEE International Conference on Advanced Information Networking and Applications, AINA*, pages 761–768, 2015.
13. Rania Ben Halima, Imen Zouaghi, Slim Kallel, Walid Gaaloul, and Mohamed Jmaiel. Formal verification of temporal constraints in business processes and allocated cloud resources. In *Proceedings of the 32nd IEEE International Conference on Advanced Information Networking and Applications, AINA*, 2018.
14. Slim Kallel, Zakaria Maamar, Mohamed Sellami, Noura Faci, Ahmed Ben Arab, Walid Gaaloul, and Thar Baker. Restriction-based Fragmentation of Business Processes over the Cloud. *Concurrency and Computation: Practice and Experience*, 2019.
15. Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals OR*, 186(1):443–464, 2011.
16. Z. Maamar, B. Thar, N. Faci, E. Ugljanin, M. Al Khafajiy, and V. Burégio. Towards a Seamless Coordination of Cloud and Fog: Illustration through the Internet-of-Things. In *Proceedings of the 34th ACM/SIGAPP Symposium On Applied Computing (SAC'2019)*, Limassol, Cyprus, 2019.
17. Madhavan Mukund. Linear-time temporal logic and bchi automata, 1997.
18. Lina Ni, Jinquan Zhang, Changjun Jiang, Chungang Yan, and Kan Yu. Resource allocation strategy in fog computing based on priced timed petri nets. *IEEE Internet of Things Journal*, 4(5):1216–1228, Oct 2017.
19. Anass Rachdi, Abdeslam En-Nouaary, and Mohamed Dahchour. Liveness and reachability analysis of BPMN process models. *CIT*, 24(2):195–207, 2016.
20. Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 228–235. IEEE, 2010.
21. Kenji Watahiki, Fuyuki Ishikawa, and Kunihiko Hiraishi. Formal verification of business processes with temporal and resource constraints. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Anchorage, Alaska, USA, October 9-12, 2011*, pages 1173–1180, 2011.
22. Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012.
23. Xun Xu. From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28(1):75 – 86, 2012.

Saoussen Cheikhrouhou obtained her diploma of engineering in 2009 and master degree in computer science in 2010 and her Ph.D. in 2015 from National School of Engineering of Sfax (University of Sfax, Tunisia). She joined the Faculty of Economics and Management (Tunisia) as Associate Professor of Computer Science in 2015. Her research interests include the business process management field and Time-aware Business Processes. More

details are available on her home page:
http://www.redcad.org/members/saoussen.cheikhrouhou/

Slim Kallel obtained his diploma of engineering and masters degree in computer science from National Engineering School of Sfax (University of Sfax, Tunisia) in 2005 and his Ph.D. from Darmstadt University of Technology (Germany) in 2011. He joined the University of Sfax as Assistant Professor of Computer Science in 2009. He became an Associate Professor in 2012. His work focused on the specification and the implementation of Time-aware business process, and adaptive systems.

Ikbel Guidara is Assistant Professor at Claude Bernard University of Lyon 1 and member of SOC research team at LIRIS-CNRS Lyon-France. Her research interests include Service-Oriented Computing, Business Process, Quality-of-Service (QoS) driven service selection and Internet of Things.

Zakaria Maamar is a Professor in the College of Technological Innovation at Zayed University, Dubai, UAE. His research interests include Internet-of-Things, social computing, and business process management. Zakaria has extensively published in different peer reviewed journals and conferences, regularly serves on the program and organizing committees of several international conferences and workshops. He also serves on the editorial boards of many international journals.

Received: April 30, 2019; Accepted: September 12, 2019.