Time-based Evaluation of Service-based Business Process Elasticity in the Cloud

Mourad Amziani∗†, Kaïs Kliai∗‡, Tarek Melliti†, Samir Tata∗

∗Institut Mines-Telecom, TELECOM SudParis, UMR CNRS Samovar, Evry, France
†IBISC, University of Evry Val d’Essonne, Evry, France
‡LIPN, CNRS UMR 7030, University of Paris 13, Villetaneuse, France

Abstract—Cloud environments are being increasingly used for the deployment and execution of service-based business processes (SBPs). Among other properties, cloud provides elasticity in order to guarantee provisioning of necessary resources that ensure a smooth functioning of cloud services despite changes in solicitations. Provisioning of elastic infrastructures and platforms is not sufficient to provide elasticity of the deployed SBP. Therefore, SBPs should be provided with elasticity mechanisms to ensure their adaptation to the workload changes. In this paper, we propose a formal model, using timed Petri nets, for SBPs elasticity considering temporal constraints and an approach for time-based evaluation of SBPs elasticity strategies.

I. INTRODUCTION

Cloud computing is a new model, based on the pay-as-you-go business principle, for provisioning of dynamically scalable and often virtualized IT services. Several types of services are delivered at different levels: infrastructure, platform, software, etc. These services use cloud components (such as databases, containers, VMs) which themselves use cloud resources (such as CPU, memory, network).

One of the most important properties that cloud environments provide is the elasticity at different levels. The principle of elasticity is to ensure the provisioning of necessary and sufficient resources such that a cloud service continues running smoothly even as the workload scales up or down, thereby avoiding under-utilization and over-utilization of resources [11].

Provisioning of resources can be made using vertical or horizontal elasticity [19]. Vertical elasticity increases or decreases the resources of a specific cloud service while the horizontal elasticity replicates or removes instances of cloud services [17]. Our work is mainly concerned with providing horizontal elasticity for Service-based Business Processes (SBPs). Cloud environments are increasingly being used for deploying and executing business processes and particularly SBPs. One of the expected facilities of Cloud environments is elasticity at the service and process levels.

It is obvious that provisioning of elastic platforms, e.g. based on elasticity of process engines or service containers [23], is not sufficient to provide elasticity of the deployed business process. Therefore, SBPs should be provided with elasticity so that they would be able to adapt to the workload changes while ensuring the desired functional and non-functional properties.

Performing elasticity consists of providing Cloud environments with mechanisms that allow deployed SBPs to scale up or down. To scale up a SBP, these mechanisms have to create, as many copies as necessary, of some business services (part of the considered SBP). To scale down a SBP, they have to remove unnecessary copies of some services. In this paper, we define two operations (duplication/consolidation) that operate at both process and service levels to ensure SBPs elasticity.

Many strategies that decide on when and how to use these mechanisms can be proposed. They use the load in each business service, in terms of the number of current invocations, as a metric to make elasticity decisions. Some of them are reactive and some others are predictive. In this paper, we define also a framework for the definition and evaluation of SBPs elasticity strategies.

There are two main approaches for describing the elasticity of SBPs. For a given SBP model, the first approach consists of producing a model for an elastic SBP which is the result of the composition of the SBP model with models of mechanisms for elasticity. This approach dedicates a controller for each deployed SBP but changes the nature of these latter. The second approach that we adopt in this paper consists in setting up a controller that enforces elasticity of deployed SBPs. One can assign a single controller for all deployed processes, a controller for each subset (that corresponds to an enterprise) or even a controller for each deployed process. Actually, we have introduced a generic controller for stateless SBP elasticity [1]. In addition, we have formally described the controller and shown how it is used to perform workload-based and resource-based evaluation of stateless [2] and stateful [3] SBPs elasticity strategies.

Nevertheless, these approaches are based on the resource consumption and the workload as metrics to evaluate SBP elasticity strategies and do not consider the temporal aspect (processing times). This can result in misinterpreting the evaluation’s results. To resolve this, we propose in this paper to go further in considering temporal constraints in modeling SBPs elasticity. In addition, we provide an approach for time-based evaluation of SBPs elasticity strategies.

The rest of this paper is organized as follows. In section II
we propose a formal model for SBP elasticity which includes temporal constraints. In section III we propose a framework for the definition and evaluation of elasticity strategies. In section IV we propose an approach for time-based evaluation of SBP elasticity. An example, for a proof of concept, is also detailed. Section V presents the state of the art and Section VI concludes and suggests some future work.

II. FORMAL MODEL FOR SBPS ELASTICITY

A SBP is a business process that consists of assembling a set of elementary IT-enabled services. These services carry out the business activities of the considered SBP. Assembling services into a SBP can be ensured using any appropriate service composition specifications (e.g. BPEL).

Elasticity of a SBP is the ability to duplicate or consolidate as many instances of the process or some of its services as needed to handle the dynamics of the received requests. Indeed, we believe that handling elasticity does not only operate at the process level but it should operate at the level of services too. It is not going to be helpful to duplicate or consolidate all the services of a considered SBP if the bottleneck comes from some services of the SBP.

A. SBP Modeling

In order to have information about temporal properties on the SBP execution, we model SBPs using Timed Petri Nets (TdPN) [15]. Classical Petri nets do not allow the modeling of time. TdPN are Petri nets where tokens are annotated with a real value that represents the age of tokens that are currently at a given place. The set of all markings over N is denoted by M(N).

Definition 2: (TdPN Marking) Let N be a Timed Petri Net. A marking M on N is a function M : P → B(R≥0) that associates to each place a finite multiset of nonnegative real numbers that represent the age of tokens that are currently at a given place. The set of all markings over N is denoted by M(N).

Definition 3: (TdPN system) A Timed Petri Net system (TdPN system) is a pair S = ⟨N, M⟩ where N is a TdPN and M a marking. The marking of a TdPN represents a distribution of calls and the age of these calls over the set of services that compose the SBP. A TdPN system models a particular distribution of calls over the services of a deployed SBP. A marked TdPN is a net system S = ⟨N, M⟩ where N is a TdPN and M is an initial marking on N where all tokens have the age 0.

Definition 4: Given a net system S = ⟨N, M⟩ and a transition t, we say that t is fireable in the marking M by tokens v = {(p, x)p, I_p, t) ∈ ∗ t ∧ |v| = |t|}, noted by M[t]∗, if:

- For all input arcs there is a token in the input place with an age satisfying the age guard of the arc, i.e. ∃(p, x) ∈ v, n_p ∈ I_p where (p, x) refers to a token in the place p of age x and I_p is the interval of the arc between p and t.

As a consequence to this restriction:...

- If the age of the token matches a value of the time interval associated to an arc connecting its place to a transition, the token can be used to fire the transition. Nevertheless, the transition is not forced to fire unless the age of the considered token reaches the upper limit of the time interval (urgency in firing).

- If the age of the token is lower than the lower limit of the time interval associated to an arc connecting its place to a transition, the token cannot be used to fire the transition and must wait for its age to increase.

- If the age of the token is higher than the upper limit of the time interval associated to an arc connecting its place to a transition, the token will never be used to fire the transition. If the token can never be used to fire
any transition, it will be considered as an outdated and unusable token.

Definition 5: Let \( M \) be a marking and \( t \) a transition, with \( M(t) = v \) for some \( v \). The firing of the transition \( t \) in \( M \), noted by \( M(t)^v M' \), changes the marking to \( M' \) s.t.
\[
M' = \begin{cases} 
M(p) \setminus \{x_r\} & \forall p \in^* t \\
M(p) \cup \{0\} & \forall p \in^* t \\
M(p) & \forall p \notin (t \cup t^*)
\end{cases}
\]

Where \( \setminus \) and \( \cup \) are operations on multisets. Note that the firing of transitions does not cause aging of tokens.

B. Elasticity Operations

Place Duplication:

Definition 6: Let \( S = \langle N, M \rangle \) be a TdPN system and let \( p \in P \), the duplication of \( p \) in \( S \) by a new place \( p^c \) (not \( P \)), noted as \( D(S, p, p^c) \), is a new TdPN system \( S' = \langle N', M' \rangle \) s.t.

\[ P' = P \cup \{p^c\} \]
\[ T' = T \cup T'' \text{ with } T'' = \{t' \mid t \in (p \cup p^c) \land \eta(t) = \eta(t')\} \text{ (\( \eta(t) \) generates a new copy of \( t \) which is not in \( T \)).} \]
\[ \text{Pre}'(\text{resp. Post}'): P' \times T' \text{ (resp. } T' \times P') \]
\[ \equiv P' \subseteq P' \times P' \text{ with } \equiv P^c = \equiv P \cup \{(p, p^c)\}. \text{ The place } p \text{ and its copy are equivalent.} \]
\[ \equiv T' \subseteq T' \times T' \text{ with } \equiv T' = \equiv T \cup \{(t, t') \mid t' \in T''\}. \text{ Each transition is equivalent to its copy.} \]
\[ M' : P' \rightarrow B(\mathbb{R}_{>0}) \text{ with } M'(p') = M(p) \text{ if } p' \neq p^c \text{ and } \emptyset \text{ otherwise.} \]

The \( \text{Pre}' \) (resp. \( \text{Post}' \)) functions are obtained by extending the \( \text{Pre} \) (resp. \( \text{Post} \)) to the new added places and transitions as follows:
\[
\text{Pre}'(p', I, t') = \begin{cases} 
\text{Pre}(p', I, t') & p' \in P \land t' \in T \\
\text{Pre}(p', I, t) & t \in T \land t' \in (T'' \setminus T) \\
\text{Pre}(p, I, t) & t \in T \land t' \in (T' \setminus T) \\
\emptyset & \text{otherwise.}
\end{cases}
\]
\[ \text{Post}' \text{ can be obtained by replacing } \text{Pre} \text{ by } \text{Post}. \]

Place Consolidation:

Definition 7: Let \( S = \langle N, M \rangle \) be a TdPN system and let \( p, p^c \) be two places in \( N \) with \( (p, p^c) \in \equiv P \land p \neq p^c \), the consolidation of \( p^c \) in \( p \), noted as \( C(S, p, p^c) \), is a new TdPN system \( S' = \langle N', M' \rangle \) s.t.

\[ N': \text{ the net } N \text{ after removing the place } p^c \text{ and the transitions } (p^c)^* \cup^* p^c \]
\[ M' : P' \rightarrow B(\mathbb{R}_{>0}) \text{ with } M'(p) = M(p) + M(p^c) \text{ and } M'(p') = M(p') \text{ if } p' \neq p. \]

Example 1: Figure 1-(a) represents a SBP model (empty marking) composed of 4 services. Figure 1-(b) is the resulting system from the duplication of the service \( s2_1 \) in (a), \( D((a), s2_1, s2_2) \). Figure 1-(c) is the consolidation of the service \( s2_1 \) in its copy \( s2_2 \), \( C((b), s2_2, s2_1) \).

We define the following notations to represent the four types of actions that can be performed during the execution of the SBP model:

\[ T(x) \]: action for the elapse of \( x \) unit of time.
\[ x^R(t) \]: action that fire \( x \) time the transition \( t \).
\[ D(s) \]: action that duplicate the service \( s \).
\[ C(s, s') \]: action that consolidates the service \( s' \) in its copy \( s \).

A possible execution of the SBP of Figure 1-(a) with an initial marking \( M_0 = (((s1_1, 0)) \) (one token with the age 0 in the place \( s1_1 \)) is given bellow:
\[ (s1_1, 0) \xrightarrow{T(2)} (s1_1, 2) \]
\[ (s1_1, 2) \xrightarrow{1^R(s1_1)} (s2_1, 0), (s3_1, 0) \]
\[ (s2_1, 0), (s3_1, 0) \xrightarrow{T(2)} (s2_1, 2), (s3_1, 2) \]
\[ (s2_1, 2), (s3_1, 2) \xrightarrow{1^R(s2_1)} (s3_1, 0) \]
\[ (s3_1, 0) \xrightarrow{T(3)} (s3_1, 3) \]
\[ (s3_1, 3) \xrightarrow{1^R(s3_1)} () \]

III. FRAMEWORK FOR THE DEFINITION AND EVALUATION OF SBPs ELASTICITY

Duplication and consolidation operations have been defined to allow structural adaptation of SBPs (creation and deletion of copies of services) in order to ensure elasticity. To use these mechanisms, elasticity strategies must be defined. The elasticity strategy is responsible of making decisions on the execution of elasticity mechanisms i.e., deciding when and how to use these mechanisms. The abundance of possible strategies [12], [10], [22] requires their evaluation and validation before using them to guarantee their effectiveness. In this context, we proposed in our
previous work a generic controller for workload-based and resource-based evaluation of stateless [2] and stateful [3] SBPs elasticity strategies. This generic controller was able to perform three actions in order to ensure SBPs elasticity (Routing, Duplication and Consolidation).

In this work, we model SBPs using timed Petri nets (TdPN) in order to perform time-based evaluations of SBPs elasticity strategies. Nevertheless the generic controller is not compatible with our TdPN model since it does not take into account the time dimension. For this reason, we propose to extend our generic controller with a fourth action (Time elapse) to be able to manage TdPN model. Our generic controller will have the capability to perform four actions:

- Time elapse: Is about the elapse of time on all tokens (calls) over the services of the considered SBP.
- Routing: Is about the way a load of services is routed over the set of their copies. It determines under which condition we transfer a call. We can think of routing as a way to define a strategy to control the flow of the load e.g., transfer a call iff the resulting marking does not violate the capacity of the services.
- Duplication: Is about the creation of a new copy of an overloaded service in order to meet its workload.
- Consolidation: Is about the deletion of an unnecessary copy of a service in order to meet its workload decrease.

We model our generic controller using high-level Petri nets (HLPN) [16]. As classic Petri nets, HLPN is a place-transition bipartite graph where the places are typed, the arcs are labeled and the transitions can be guarded by a condition. A general overview of our generic controller is shown in Figure 2. The controller contains one place (SBP) of type TdPN system. The marking of this place evolves either by calls transfer (Routing), by the elapse of time (Time elapse), by the duplication of an overloaded service (Duplication) or by the consolidation of an underused service (Consolidation). These four transitions (actions) are guarded by conditions that decide when and how to perform these actions. In our controller, the conditions (Ready_D, Ready_C, Ready_R) are generic to allow the use of different elasticity strategies while the condition Ready_T is implemented to increment time. By instantiating our generic controller, one can analyze and evaluate time-based behaviors and performances of the implemented strategies.

IV. TIME-BASED EVALUATION OF SBPs ELASTICITY

In this section we will show how we can use the generic controller to perform time-based evaluations of elasticity strategies. To do so, we have to instantiate the controller with the elasticity strategies we want to evaluate and then generate its reachability graph. This graph will contain all the possible evolutions of the SBP with respect to time constraints, calls arrival and implemented strategies. Using this reachability graph, many properties can be checked and many indicators can be calculated according to the evolution of the controller i.e., each state of the reachability graph will contain the values of the indicators. In this work we are interested in performing time-based evaluation of elasticity strategies. For this, we define two indicators:

- Call processing time: The time required for the processing of calls received by the SBP. This time depends on the SBP (time constraints, service capacities), on the elasticity strategy and on the calls arrival. We calculate the time required for the processing of each received call. This indicator provides us with information about the time necessary for SBP calls processing. With this indicator we can observe the influence of the implemented strategy on the SBP processing time according to the evolution of calls arrival.
- Loss of calls: As explained previously, it is possible to have loss of SBP calls during the SBP execution. This loss is due to the outdated calls that can be caused by the temporal constraints on the SBP during its execution (expired calls that will never be treated). We calculate for each execution, the number of lost calls according to calls arrival. This indicator provides us with information about the proportion of lost calls. With this indicator we can observe the influence of the implemented strategy on the proportion of lost calls according to the evolution of calls arrival.

For the proof of concept, we present hereafter an example of time-based evaluation of elasticity strategies with our controller. The controller is implemented using the SNAKES toolkit. SNAKES is a Python library that allows the use of arbitrary Python objects as tokens and arbitrary Python expressions in transitions guards, etc [18]. Note that in our implementation we consider only natural numbers as extremes of the time intervals.

1) Experimental Setup: We propose here to instantiate our generic controller with four elasticity strategies. Two strategies are inspired from the literature [14], [6] and two strategies are defined by us in order to illustrate the feasibility of our evaluation approach. For each implemented strategy we execute our instantiated controller with the SBP system of Figure 1-(a) and we calculate the indicators
defined previously (time processing and lost calls).

We assume in this example that each service of the SBP is provided with a maximum and minimum threshold capacities. Above the maximum threshold the QoS would no longer be guaranteed and under the minimum threshold we have an over allocation of resources. Here are the thresholds:

- \( \text{Max}_t(s_1 \_1) = 20, \text{Min}_t(s_1 \_1) = 1 \)
- \( \text{Max}_t(s_2 \_1) = 3, \text{Min}_t(s_2 \_1) = 1 \)
- \( \text{Max}_t(s_3 \_1) = 15, \text{Min}_t(s_3 \_1) = 1 \)
- \( \text{Max}_t(s_4 \_1) = 15, \text{Min}_t(s_4 \_1) = 1 \)

Note here that these thresholds represent the maximum number of running instances (calls) on each service. These thresholds are used as scaling indicators by the strategies in order to make their elasticity decisions.

In our experiment, we want to observe how the load variation affects the call processing time on the SBP. The load variation represents the variation in the number of SBP invocation. An invocation (a call) of the SBP is represented by adding a token to a copy of the place \( s_1 \_1 \), the invocation ends by removing a token from a copy of the place \( s_4 \_1 \).

While the duplication is a time-consuming operation (time required to create a new copy of a service), the consolidation is not (no consumption of time for the removal of an empty copy of a service). We assume in this example that the temporal costs of duplication/consolidation are as follows: \( T_{\text{dup}} = 1 \) and \( T_{\text{con}} = 0 \).

2) Elasticity Strategies: As we explained previously, the definition of a strategy consists of instantiating the three generic predicates \( \text{ready}_R \), \( \text{ready}_D \) and \( \text{ready}_C \) (the predicate \( \text{ready}_T \) is already instantiated to increment the age of SBP tokens). In our experiments we use threshold-based scaling algorithms that use the concept of maximum and minimum thresholds to make elasticity decisions. Hereafter the strategies:

Strategy 1: In [14] an algorithm is proposed to scale up or down an application instance by replication in response to a change in the workload.

- \( \text{Ready}_D(S, s) : M(s) \geq \text{Max}_t(s) \land \exists s' \in [s] : M(s') < \text{Max}_t(s') \land \forall t \in* [s] : M[t] \). It duplicates a service \( s \) if all copies of this service have already reached their maximal threshold. In addition, there is a call waiting to be transferred to this service \( s \).
- \( \text{Ready}_C(S, s', s) : M(s') = 0 \land M(s) \leq \text{Min}_t(s') \land \exists t \in* [s] : M[t] \). It consolidates a copy \( s' \) of service if this copy does not contain calls (empty copy) and there is another copy \( s \) of the service that has not reached its minimum threshold. In addition, there is no call waiting to be transferred to this copy \( s \).
- \( \text{Ready}_R(S, t) : \forall s \in P : M'(s) < \text{Max}_t(s) \) with \( M[t], M' \). It routes a call if this call transfer does not cause a violation of the maximum thresholds of services.

Strategy 2: In [6] a scaling algorithm is proposed to scale up or down the number of instances according to a threshold in each instance.

- \( \text{Ready}_D(S, s) : M(s) \geq \text{Max}_t(s) \land \exists s' \in [s] : M(s') < \text{Max}_t(s') \). It duplicates a copy \( s \) of service if all copies of this service have already reached their maximal threshold.
- \( \text{Ready}_C(S, s', s) : M(s') = 0 \land M(s) \leq \text{Min}_t(s) \).
- \( \text{Ready}_R(S, t) : \forall s \in P : M'(s) < \text{Max}_t(s) \) with \( M[t], M' \). Same routing strategy as strategy 1.

Strategy 3: We define a third strategy that implements only a routing strategy.

- \( \text{Ready}_R(S, t) : \forall s \in P : M'(s) < \text{Max}_t(s) \) with \( M[t], M' \). Same routing strategy as strategies 1 and 2.

Strategy 4: We define also a fourth strategy that implements an algorithm which takes its elasticity decision depending on the load of services and the way the load can be transferred between services.

- \( \text{Ready}_D(S, s) : \exists s' \in* (s) : M([s']) > \text{Max}_t([s]) - M([s]). \)
- \( \text{Ready}_C(S, s', s) : M(s') = 0 \land M(s) \leq \text{Min}_t(s) \land \exists s'' \in* (s') : M([s'']) > \text{Max}_t([s]) - M([s]). \)
- \( \text{Ready}_R(S, t) : \forall s \in P : M'(s) < \text{Max}_t(s) \) with \( M[t], M' \). Same routing strategy as all other strategies.

3) Evaluation of Strategies: Figure 3 represents the evolution of the minimal calls processing time according to load variations. The Figure 4 represents the evolution of the average calls processing time according to load variations. Figure 5 represents the evolution of the proportion of lost calls according to load variations.

From these Figures we can differentiate two phases:

- We observe at the beginning of the execution (load: 0-3 calls) that the number of calls does not create an overload on services. In this case, we can see that the SBP reacts the same way with the four strategies (in terms of processing time and proportion of lost calls). This is explained by the fact that no duplication/consolidation is needed to handle the calls arrival. The four strategies are going to use only their routing strategy.
- We observe that from a load value greater than 3 calls (load: 4-15 calls) the number of calls creates an
overload on the service $s_{2\_1}$ ($\max_{t}(s2) = 3$). In this case, we can see that the process reacts in a different way according to the implemented strategy. In fact, the strategy 3 does not allow duplication of the overloaded service causing an important loss of calls. Strategies 1 and 2 react only when the service $s_{2\_1}$ becomes overloaded by duplicating it. We can see that these strategies are not suitable for ensuring the elasticity of SBPs with temporal constraints because of their late reactions. This delay in the reaction can cause a loss of calls. Strategy 4 prevents the overload of the service $s_{2\_1}$ by duplicating it when the load of the service $s_{1\_1}$ (the service that will transfer its calls to the service $s_{2\_1}$) may cause the overload of service $s_{2\_1}$ when its load will be transferred to service $s_{2\_1}$. We can see that this strategy is more suitable for ensuring the elasticity of the considered SBPs than the other strategies.

We can observe that the SBP processing time with the strategy 3 is constant from a load greater than 3. We explain this by the fact that the strategy will only treat the first calls and will lose all other calls (which explains the high proportion of loss). We can also observe that the processing time with the strategy 4 is higher than the other strategies (especially with highest loads). We explain this by the fact that this strategy is the most reliable, which means that it successfully treats more calls compared to other strategies thereby increasing the overall processing time.

Hereafter, we give some possible executions of the SBP with the four strategies. These executions result from an initial marking $M_0 = 4'(s_{1\_1},0)$

**Strategy 3:** This strategy does not implement duplication/consolidation mechanisms. So, when the service $s_{2\_1}$ reached its maximum load threshold ($\max_{t}(s2) = 3$) we cannot transfer a call to this service. It is necessary in this case to wait until the service is done with one of its current calls in order to receive a new one. Nevertheless, in the considered SBP if we wait until the service $s_{2\_1}$ finishes processing one of its calls and transfer it to another service ($s_{3\_1}$), the call waiting to be transferred to the service $s_{2\_1}$ will be expired (outdated call). Hereafter we give a possible execution of the SBP with strategy 3:

- $4'(s_{1\_1},0) \xrightarrow{T(2)} 4'(s_{1\_1},2)$
- $4'(s_{1\_1},2) \xrightarrow{3'R(t_{1\_1})} 3'(s_{2\_1},0), 3'(s_{3\_1},0), (s_{1\_1},2)$
- $3'(s_{2\_1},0), 3'(s_{3\_1},0), (s_{1\_1},2) \xrightarrow{T(2)} 3'(s_{2\_1},2), 3'(s_{3\_1},2), (s_{1\_1},4)$
- $3'(s_{2\_1},2), 3'(s_{3\_1},2) \xrightarrow{3'R(t_{2\_1})} 3'(s_{3\_1},0)$
- $3'(s_{3\_1},0) \xrightarrow{T(3)} 3'(s_{3\_1},3)$
- $3'(s_{3\_1},3) \xrightarrow{3'R(t_{3\_1})} ()$

We can see in this execution that we lost a call which represents 25% of lost calls.

**Strategies 1 and 2:** Both strategies use duplication/consolidation mechanisms that react when a service is overloaded. In the considered SBP, when the service $s_{2\_1}$ reached its maximum load threshold ($\max_{t}(s2) = 3$) both strategies duplicate it. The duplication of service $s_{2\_1}$ can generate two types of execution (favorable or unfavorable execution). The favorable execution solves the problem of overloaded service $s_{2\_1}$ while ensuring the treatment of all calls. Whereas, the unfavorable execution results in the loss of calls in spite of the duplication of the overloaded service.

Hereafter we give two possible executions (a favorable and an unfavorable one) of the SBP with the strategies 1 and 2. The favorable possible execution:

- $4'(s_{1\_1},0) \xrightarrow{T(2)} 4'(s_{1\_1},2)$
we give a possible execution with the strategy 4:

- 4′(s1_1, 2) \xrightarrow{\pi'(s1_1,1)} 3′(s2_1, 0), 3′(s3_1, 0), (s1_1, 2)
- 3′(s2_1, 0), 3′(s3_1, 0), (s1_1, 2) \xrightarrow{D(s2_1)} 3′(s2_1, 1), 3′(s3_1, 1), (s1_1, 3)
- 3′(s2_1, 1), 3′(s3_1, 1), (s1_1, 3) \xrightarrow{1′R(t1_1)} 3′(s2_1, 1), (s3_1, 1), (s2_2, 0), (s3_1, 0)
- 3′(s2_1, 1), 3′(s3_1, 1), (s2_2, 0), (s3_1, 0) \xrightarrow{T(1)} 3′(s2_1, 2), 3′(s3_1, 1), (s2_2, 1), (s3_1, 1)
- 3′(s2_1, 2), 3′(s3_1, 1), (s2_2, 1), (s3_1, 1) \xrightarrow{3′R(t2_1)} 3′(s3_1, 0), (s2_2, 1), (s3_1, 1)
- 3′(s3_1, 0), (s2_2, 1), (s3_1, 1) \xrightarrow{T(1)} 3′(s3_1, 1), (s2_2, 2), (s3_1, 2)
- 3′(s3_1, 1), (s2_2, 2), (s3_1, 2) \xrightarrow{1′R(t2_2)} 3′(s3_1, 1), (s3_1, 0)
- 3′(s3_1, 1), (s3_1, 0) \xrightarrow{C(s2_2, s2_1)} 3′(s3_1, 1), (s3_1, 0)
- 3′(s3_1, 1), (s3_1, 0) \xrightarrow{T(2)} 3′(s3_1, 1), (s3_1, 2)
- 3′(s3_1, 1), (s3_1, 2) \xrightarrow{T(1)} 3′(s3_1, 3)
- (s3_1, 3) \xrightarrow{1′R(t3_1)} ()

The unfavorable possible execution:

- 4′(s1_1, 0) \xrightarrow{T(3)} 4′(s1_1, 3)
- 4′(s1_1, 3) \xrightarrow{3′R(t1_1)} 3′(s2_1, 0), 3′(s3_1, 0), (s1_1, 3)
- 3′(s2_1, 0), 3′(s3_1, 0), (s1_1, 3) \xrightarrow{D(s2_1)} 3′(s2_1, 1), 3′(s3_1, 1), (s1_1, 4)
- 3′(s2_1, 1), 3′(s3_1, 1), (s1_1, 3) \xrightarrow{T(1)} 3′(s2_1, 2), 3′(s3_1, 1, 2)
- 3′(s2_1, 2), 3′(s3_1, 1, 2) \xrightarrow{3′R(t2_1)} 3′(s3_1, 0)
- 3′(s3_1, 0) \xrightarrow{C(s2_2, s2_1)} 3′(s3_1, 1, 0)
- 3′(s3_1, 0) \xrightarrow{T(3)} 3′(s3_1, 1, 3)
- 3′(s3_1, 1, 3) \xrightarrow{3′R(t3_1)} ()

V. RELATED WORK

Elasticity is the ability to determine the amount of resources to be allocated as efficiently as possible according to user requests. Many predictive or reactive strategies have been proposed to address this issue [10], [22]. Reactive strategies [6], [14], [5] are based on Rule-Condition-Action mechanisms. While predictive strategies [20], [9] are based on predictive-performance models and load forecasts.

At the Infrastructure level, generally two approaches are used to perform elasticity: Vertical elasticity which consists of adding or removing resources to virtual machines (VMs) to prevent over-loading and under-loading [7], [20], [9]. Horizontal elasticity on the other hand consists of adding or removing instances of VMs according to demands variations [13], [4], [9]. These approaches ensure the elasticity at the infrastructure level, but they are not sufficient to ensure the elasticity of deployed process. At the platform level, elasticity mechanisms have been proposed to ensure the elasticity of deployed processes. Nonetheless, provisioning of elastic platforms is not sufficient to ensure elasticity of deployed SBPs since they do not take into account the nature of the application e.g., SBPs.

At the Software level, SBPs mechanisms must be provided to ensure the elasticity of SBPs. In [8], the authors propose an approach to ensure elasticity of processes in the Cloud by adapting resources and their non-functional properties with respect to quality and cost criteria. Nevertheless, the authors addressed elasticity of applications in general rather than processes particularly. In [21], the authors consider scaling at both the service and application levels in order to ensure elasticity. They discuss the elasticity at the service level as we did in our approach. Nevertheless, the proposed approach is not based on a formal model. In [17], the authors present ElasS, a service implemented as a SaaS application for managing elasticity in the Cloud. While the idea of pushing elasticity management to the applications is in line with our approach, the proposed approach is difficult to use since it
requires an effort from the application designer to provide the necessary information for elasticity enforcement.

In [1] we considered the elasticity of stateless SBPs and provided a duplication and consolidation mechanisms. In [2] we formally proved the correctness of our elasticity mechanisms. In addition, we have provided a framework to evaluate strategies based on duplication/consolidation. In [3] we considered the elasticity of stateful SBPs. In this work we went further by considering temporal constraints on SBPs elasticity. We also proposed an approach for time-based evaluation of elasticity strategies.

VI. CONCLUSION

In this paper, we proposed a time-based evaluation of SBPs elasticity strategies. We model SBPs using timed Petri nets and formalized elasticity mechanisms that operate at both process and service levels in order to ensure SBPs elasticity while considering temporal constraints. In addition, we have proposed an approach for time-based evaluation of elasticity strategies and we presented an example for the proof of concept. Actually, we are using strategies that are based on the load in services, in terms of the number of current invocations, as a metric to make elasticity decisions. In our future work, we plan to implement strategies that are based on time (service processing time, temporal constraints) as metric to make elasticity decisions (time-based strategies). We consider also the implementation of the elasticity operations into Cloud.

VII. ACKNOWLEDGEMENTS

The work presented in this paper was partially funded by the French FUI CompatibileOne, the French FSN OpenPaaS and the European ITEA Easi-Clouds projects.

REFERENCES


