Formal Modeling and Evaluation of Stateful Service-based Business Process Elasticity in the Cloud

Mourad Amziani[1,2], Tarek Melliti[2], and Samir Tata[1⋆]

[1] Institut Mines-Telecom, TELECOM SudParis, UMR CNRS Samovar, Evry, France
[2] University of Evry Val d'Essonne, IBISC, Evry, France

Abstract. Cloud environments are being increasingly used for deploying and executing business processes and particularly Service-based Business Processes (SBPs). One of the expected features of Cloud environments is elasticity at different levels. It is obvious that provisioning of elastic platforms 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. In this paper, we propose a formal model for stateful SBPs elasticity that features a duplication/consolidation mechanisms and a generic controller to define and evaluate elasticity strategies.

Key words: Cloud computing, stateful service-based business processes, elasticity, evaluation of elasticity strategies

1 Introduction

Based on the pay-as-you-go business principle, the Cloud computing is a new model 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 etc.) which themselves use cloud resources (such as CPU, memory, network).

Among other properties cloud environments provide elasticity. 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 number or quantity of its use scales up or down, thereby avoiding under-utilization and over-utilization of resources [10].

Provisioning of resources can be made using vertical or horizontal elasticity [17]. Vertical elasticity increases or decreases the resources of a specific cloud service while the horizontal elasticity replicates or removes instances of cloud services [15]. Our work is mainly concerned with providing horizontal elasticity for Services-based Business Processes (SBPs). This paper does not discuss all the

⋆ The work presented in this paper was partially supported by the OpenPaaS project.

aspects that are relevant to elasticity. For example, we do not deal with ensuring vertical elasticity of cloud services and infrastructure services. While we believe that these issues are important, the provisioning of horizontal elasticity of SBPs discussed here is complex enough in itself to deserve separate treatment.

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 [21], 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.

In this paper we address elasticity at the level of SBPs that mainly raises the following questions.

- What mechanisms should be developed to perform elasticity of SBPs?
- How to define and evaluate elasticity strategies of SBPs?

Among others, there are two main approaches for describing elasticity of SBPs. For a given SBP model, the first approach consists in 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 SBP deployed 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 the elasticity of business processes based on stateless services [1]. In addition, we have formally described the controller and shown how it is used for the evaluation of elasticity strategies [2]. In this paper we go further in considering the elasticity of stateful SBPs. In addition, we provide two approaches for the evaluation of elasticity strategies.

Many strategies that decide on when SBP elasticity is performed can be proposed. They use the load in each business service, in terms of the 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 propose formal descriptions and an evaluation framework of reactive strategies.

The rest of this paper is organized as follows. Section 2 presents the state of the art. In section 3 we propose a deployment model for SBPs. In section 4, which is dedicated to the first question we raised above, we propose a formal model for elasticity of stateful SBPs. In section 5, which is dedicated to the second question we raised above, we propose a framework for the definition and evaluation of elasticity strategies using two evaluation approaches. An example, for a proof of concept, is also detailed. Section 6 concludes and suggests directions for our future work.

2 Related Work

One of the most relevant issues raised by the Cloud environment is the elasticity at different levels. Elasticity is the ability to determine the amount of resources to be allocated as efficiently as possible according to user requests. Many approaches based on predictive or reactive strategies have been proposed to address this issue [9, 20]. Reactive strategies [5, 13, 4] are based on Rule-Condition-Action mechanisms. While predective strategies [18, 8] 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 in adding or removing resources to virtual machines (VMs) to prevent over-loading and under-loading [6, 18, 8]. Horizontal elasticity on the other hand consists of adding or removing instances of VMs according to demands variations [12, 3, 8]. 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 containers elasticity [4, 21]. Nonetheless, provisioning of elastic platforms is not sufficient to provide elasticity of deployed SBPs since they do not take into account the nature of the application *e.g.,* SBPs. In fact, each application has a maximal capacity, beyond this capacity the QoS decreases and can make the container unresponsive and consequently, crash the application. Giving to the container more resources will not solve the problem [21].

At the Software level, SBPs mechanisms must be provided to ensure the elasticity of SBPs. In [7], 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 [19], 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 [15], the authors present *ElaaS*, 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 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 this work we go further by considering the elasticity of stateful SBPs. We also propose two approaches for the evaluation of elasticity strategies.

At the best of our knowledge, the approaches for elasticity mainly those we cite above, focus on the IaaS level. As stated before, ensuring elasticity at the IaaS level is not sufficient to provide users with elasticity of deployed SBPs. Similarly, ensuring elasticity at the PaaS level is not enough to ensure elasticity of deployed SBPs. We believe that elasticity should be handled and tuned at

different levels of Cloud environments. We have already contributed to the elasticity of platforms at the PaaS level [21]. The work we present in this paper is novel in the sense that it (1) tackles the problem of elasticity at the SaaS level (particularly for stateful SBPs) and (2) is based on a formal model (3) proposes a framework for defining and evaluating elasticity strategies and (4) proposes two approaches for the evaluation of elasticity strategies.

3 Model for SBPs Deployment

A SBP is a business process that consists in 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). In Figure 1-(a) we presents an example of SBP composed by eight services modeled in BPMN.

To model SBPs, several techniques can be used (BPEL, BPMN, Petri nets). In our work, we are interested in the formal aspect of the model. So, we choose Petri nets to model SBPs. Generally the modeling of SBPs using Petri nets represents the SBPs execution model.

3.1 SBPs Execution Model

The SBPs execution model specifies how the processes and their services need to be executed and in what order. In this model, each service is represented by a transition. The places represent the states between services.

The execution model of the SBP of Figure 1-(a) gives the Petri net shown in Figure 1-(b).

The SBPs execution model is suitable to verify behavioral properties. Nevertheless, with this model we can not verify non-functional properties e.g. QoS properties. Indeed, the execution model does not provide a view of the evolution of loads on services which is necessary to verify this kind of properties. Therefore, it would be interesting to have a view of the way services are deployed and their loads. For that reason, we propose, using a transformation procedure [3], to automatically derive a deployment model from the execution model of a SBP.

3.2 SBPs Deployment Model

The obtained SBPs deployment model is also modeled using Petri nets. In this model, each service is represented by a place. The transitions represent calls transfers between services according to the behavior specification of the SBP. In fact, instead of focusing on the execution model of the process and its services, we focus on the dynamic (evolution) of loads on each basic service participating in the SBP.

[3] Due to the lack of space and the heaviness of notations, the transformation rules are not given in this paper

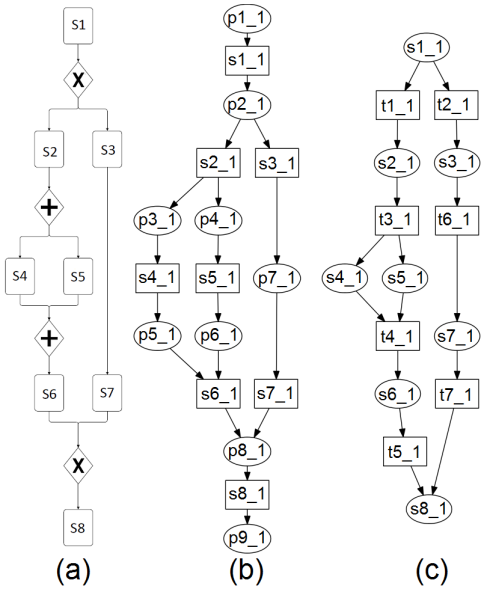

Fig. 1. An example of the transformation of a SBP execution model (b) to a deployment model (c)

The SBP deployment model of the SBP execution model of Figure 1-(b) gives the Petri net shown in Figure 1-(c).

The SBPs deployment model represents the way a process and its services are deployed and the load on each services of the SBP. The advantage of using this deployment model is to be able to represent information that are inexpressible on the execution model. This would allow verifying some properties that cannot be verified in the execution model e.g. QoS, deployment properties. Using the deployment model we can, for example, monitor the load of a service (the number of current invocations of a service) which is represented by the marking of its corresponding place. The marking of places represents load distribution over services of the process. This facilitates the implementation of load-based mechanisms e.g. elasticity and load balancing mechanisms.

In the rest of paper we will focus on the elasticity of SBPs. For that reason, we will use the deployment model to represent SBPs.

4 Formal Model for Stateful SBPs Elasticity

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 necessary to duplicate or consolidate all the services of a considered SBP while the bottleneck comes from some services of the SBP.

Services involved in a SBP can be stateless or stateful services. A stateless service is a service that does not store its state between two service invocations. Each service invocation is completely independent of previous invocations. On the other hand, a stateful service is a service designed to store its state between invocations. Interactions and events occurring during service execution are taken into account to manage the service invocations. The state of a stateful service is represented by the user sessions and the data values specific to this service.

Performing elasticity on stateless services can be done using a service duplication/consolidation approach without taking into account the state of the duplicated/consolidated service [1]. However, performing elasticity on stateful service is more complicated. In fact, in a duplication/consolidation approach it is necessary to ensure that the state of the stateful service is taken into account in the elasticity mechanisms at each duplication or consolidation. To solve this problem, we propose to model stateful SBPs using Colored Petri Nets (CPN). In our model, the management of user sessions is allowed by the use of colors. Each user session represents a state of the service, and so, represents a color. On the other hand, to model the data values specific to a stateful service, we propose to model each stateful service by a stateless service and a database deployed as a service in which the data values of the service are stored during its execution. Each stateful service of the SBP will have its specific database service that models the data values of all user sessions. Note that this database service can be also duplicated/consolidated as other services that compose the SBP in order to ensure its elasticity.

4.1 Stateful SBP Modeling

To model stateful SBP we use Colored Petri Nets (CPN). Classical Petri nets does not allow the modeling of data. CPN have been proposed to extend Petri nets by modeling data with color. A Petri net is a colored Petri net if its tokens can be distinguished by colors. Each place has an associated type determining the kind of data that this place may contain. The marking of a given place is a multi-set of values of the associated type. Arcs constraints are expressions that extract or produce multi-sets with respect to the sources of target types.

In order to give a definition of the CPN, we give here, without a loss of generality, a simple syntax and semantics for expressions.

- Types: Noted by Π, we range over by using π. Types are defined by the set of values that compose them, $\pi = \{v_0, ..., v_i, ...\}$. Also, types can be defined by applying set operations on them.
- Variables: Noted by $\mathcal{X}$, we range over by $\mathcal{X}_i$, Variables are typed and as usual we use $Type(\mathcal{X})$ to obtain the type of $\mathcal{X}$.
- Function: Denoted by F, for a function $f \in F$ with $f : \pi \rightarrow \pi'$ we use $Type(f)$ to define its range type.

Definition 1. *(Multi-set) : Let E be a set, a multi-set m on E is an application from E to $\mathbb{N}$, we write such a multi-set using the formal sum notation i.e $m = \sum_{0<i\leq|E|} q'_i e_i$ (with $q_i \in \mathbb{N}$ and $e_i \in E$)[4]. We denote by $\mathcal{M}(E)$ the set of multi-sets of E.*

We use $\mathcal{E}$ to define a color expression which can be a color constant, variable, or a color function. Given an expression $e \in \mathcal{E}$, we use $Var(e)$ to denote the set of variables which appear in e.

Definition 2. *(CPN graph) : A stateful SBP deployment model is a Colored Petri Net graph (CPN graph) $N=\langle \Sigma, P, T, cd, Pre, Post, \equiv_P, \equiv_T \rangle$, where:*

- *Σ is a set of non-empty types, also called color sets (represents the set of user sessions).*
- *P is a set of labeled places (represents the set of services/activities involved in a SBP);*
- *T is a set of labeled transitions (represents the call transfers between services according to the SBP behavioral specification);*
- *$cd : P \rightarrow \Pi$ is a function that associates to each place a color domain. Intuitively, this means that each token in place p must have a data value that belongs to $cd(p)$;*
- *Pre (resp. $Post$): are forward (resp. backward) matrices, such that $Pre : P \times T \rightarrow \mathcal{M}(\mathcal{E})$ (resp. $Post : P \times T \rightarrow \mathcal{M}(\mathcal{E})$, represent the input (resp. output) arc expressions.*
- *$\equiv_P \subseteq P \times P$: an equivalence relation over P. An equivalence relation between copies of the same place: $[p]_{\equiv_P} = \{p' | (p, p') \in \equiv_P\}$.*
- *$\equiv_T \subseteq T \times T$: an equivalence relation over T. An equivalence relation between copies of the same transition: $[t]_{\equiv_T} = \{t' | (t, t') \in \equiv_T\}$.*

In our model, each service is represented by a place with a session identifier as an associated type. Each service call is typed with its session identifier. The transitions represent calls transfers between services according to the behavior specification of the SBP while respecting the different user sessions.

As stated above, in order to manage the data values of stateful services, we add a place (database service) for each stateful service of the SBP to model the data values related to this stateful service. If the SBP contains a certain number of stateful services, we will have the same number of database services so each database service manage the data values of its corresponding stateful service. For each stateful service $s \in P$:

- *$P = P \cup \{sDB\}$ (sDB: database service of the stateful service s)*
- *$\forall t \in T : Pre(sDB, t) = Pre(s, t) \wedge Post(t, sDB) = Post(t, s)$*

For a place p and a transition t we denote ${}^\bullet p$ and $p^\bullet$ as the input and output transitions set of place p, ${}^\bullet t$ and $t^\bullet$ as the input and output places set of transition t.

[4] For simplicity we keep only the terms with $q_i \neq 0$

The $^\bullet$ notation can also be naturally extended to equivalent classes of places and/or transitions as the union of its application to all the elements of the class *e.g.* $[p]^\bullet = \bigcup_{p' \in [p]} p'^\bullet$. We extend the notation [] to a set of places and transitions *e.g.* for some $P' \subseteq P$, $[P']_{\equiv_P} = \{[p]_{\equiv_P} | p \in P'\}$. We ignore the $\equiv_T$ and $\equiv_P$ if it is clear from the context.

Definition 3. *(Well-formed graph) A CPN graph* $N = \langle \Sigma, P, T, cd, Pre, Post, \equiv_P, \equiv_T \rangle$ *is well formed iff:* $\forall t \in T, \forall p \in t^\bullet$*, we have* $Var(Post(p,t)) \subseteq Var(Pre(.,t))$ *with* $Var(Pre(.,t)) = \bigcup_{p' \in {}^\bullet t} var(Pre(p',t))$.

In a well-formed CPN graph, we restrict that for each transition, the output arc expressions must be composed by the variables which are in the input arcs expressions. To each CPN graph, we associate its terms incidence Matrix C ($P \times T \to \mathcal{M}(\mathcal{E})$) with $C = Post - Pre$.

In the following, we define the behaviors (the dynamics) of a CPN System.

Definition 4. *(CPN Marking) A marking* M *of a CPN graph is a multiset vector indexed by* P*, where* $\forall p \in P, M(p) \in \mathcal{M}(cd(p))$*. The marking is also extended to equivalent classes i.e.* $M([p]) = \sum_{p' \in [p]} M(p')$*. The marking of a CPN represents a distribution of calls over the set of services that compose the SBP.*

Definition 5. *(CPN system) A Colored Petri Net system (CPN system) is a pair* $S = \langle N, M \rangle$ *where* N *is a CPN graph and* M *is one of its marking. A CPN system models a particular distribution of calls over the services of a deployed SBP.*

We use $u : Var(Pre(.,t)) \to \Sigma$ with $M \geq Pre(.,t)^u$ to denote a binding of the input arcs variables. [5]

Definition 6. *Given a CPN system* $S = \langle N, M \rangle$ *and a transition* t*, we use* $M[t\rangle^u$ *to denote that the transition* t *is fireable in the marking* M *by the use of* u*, and we use the classic notation* $M[t\rangle$ *if* u *is not important (e.g. when* u *is unique). A class of transitions is fireable in* M*,* $M[t\rangle^u$*, iff* $\exists t' \in [t] : M[t'\rangle^u$

Definition 7. *Let* M *be a marking and* t *a transition, with* $M[t\rangle^u$ *for some* u*. The firing of the transition* t *changes the marking of CPN from* M *to* $M' = M + C(.,t)^u$*. We note the firing as* $M[t\rangle^u M'$.

The transition firing represents the evolution of the load distribution after calls transfer. The way that calls are transferred between services depends on the behavior specification (workflow operators) of the SBP.

[5] u must respect the color domain of the places, *i.e.* , $\forall p \in^\bullet t$, $x \in var(Pre(p,t))$, we have $u(x) \in cd(p)$.

4.2 Elasticity Operations

Place Duplication

Definition 8. *Let $S = \langle N, M\rangle$ be a CPN system and let $p \in P$, the duplication of p in S by a new place p^c ($\notin P$), noted as $D(S, p, p^c)$, is a new CPN system $S' = \langle N', M'\rangle$ s.t*

- $\Sigma' = \Sigma$
- $P' = P \cup \{p^c\}$
- $T' = T \cup T''$ *with* $T'' = \{t^c | t \in ({}^\bullet p \cup p^\bullet) \wedge t^c = \eta(t)\}$ *($\eta(t)$ generates a new copy of t which is not in T).*
- $cd' : P' \to \Sigma'$ *with* $cd'(p') = cd(p')$ *for all* $p' \in P$ *and* $cd'(p^c) = cd(p)$
- Pre' *(resp.* $Post'$*):* $P' \times T' \to \mathcal{M}(\mathcal{E})$ *(resp.* $P' \times T' \to \mathcal{M}(\mathcal{E})$*)*
- $\equiv_{P'} \subseteq P' \times P'$ *with* $\equiv_{P'} = \equiv_P \cup \{(p, p^c)\}$*. The place p and its copy are equivalent.*
- $\equiv_{T'} \subseteq T' \times T'$ *with* $\equiv_{T'} = \equiv_T \cup \{(t, t^c) | t^c \in T''\}$*. Each transition is equivalent to its copy.*
- $M' : P' \to \mathcal{M}(cd(p))$ *with* $M'(p') = M(p')$ *if* $p' \neq p^c$ *and* $\emptyset$ *otherwise.*

The Pre' (resp. $Post'$) functions are obtained by extending the Pre (resp. $Post$) to the new added places and transitions as follow:

$$Pre'(p', t') = \begin{cases} Pre(p', t') & p' \in P \wedge t' \in T \\ Pre(p', t) & t \in T \wedge t' \in (T' \setminus T) \wedge t' \in [t]_{\equiv_{T'}} \wedge p' \in (P \setminus \{p\}) \\ Pre(p, t) & t \in T \wedge t' \in (T' \setminus T) \wedge t' \in [t]_{\equiv_{T'}} \wedge p' = p^c \\ \emptyset & otherwise. \end{cases}$$

$$Post'(t', p') = \begin{cases} Post(t', p') & p' \in P \wedge t' \in T \\ Post(t, p') & t \in T \wedge t' \in (T' \setminus T) \wedge t' \in [t]_{\equiv_{T'}} \wedge p' \in (P \setminus \{p\}) \\ Post(t, p) & t \in T \wedge t' \in (T' \setminus T) \wedge t' \in [t]_{\equiv_{T'}} \wedge p' = p^c \\ \emptyset & otherwise. \end{cases}$$

Place Consolidation

Definition 9. *Let $S = \langle N, M\rangle$ be a CPN system and let p, p^c be two places in N with $(p, p^c) \in \equiv_P \wedge p \neq p^c$, the consolidation of p^c in p, noted as $C(S, p, p^c)$, is a new CPN system $S' = \langle N', M'\rangle$ s.t*

- N'*: is the net N after removing the place p^c and the transitions $(p^c)^\bullet \cup {}^\bullet p^c$*
- $M' : P' \to \mathcal{M}(cd(p))$ *with* $M'(p) = M(p) + M(p^c)$ *and* $M'(p') = M(p')$ *if* $p' \neq p$.

Example 1. Figure 2-(a) represents the deployment model (empty marking) of the stateful SBP of Figure 1-(a). In this SBP, $s3\_1$ is a stateful service and all others are stateless services. Figure 2-(b) is the resulting system from the duplication of the service $s3\_1$ in (a), $D((a), s3\_1, s3\_2)$. Figure 2-(c) is the consolidation of the service $s3\_1$ in its copy $s3\_2$, $C((b), s3\_2, s3\_1)$.

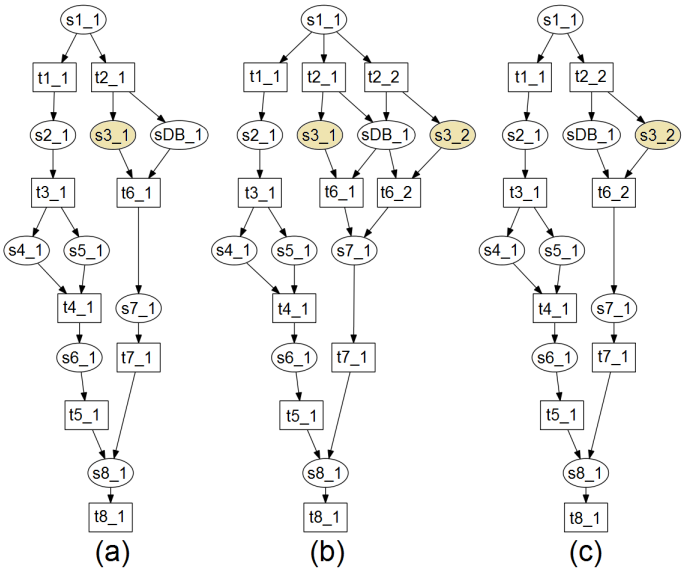

Fig. 2. An example of the elasticity of stateful SBP

4.3 Correctness of Elasticity Operations

In the previous paper [2] we applied the same structural duplication and consolidation operations on classical Petri-net. We proved that this two operations preserve the structural and dynamical properties of the net modulo $\equiv_T$ and $\equiv_P$ relations. This means that the two following properties are still valid for colored Perti-nets:

Property 1 By any transformation of the net using duplication/consolidation operators, we do not lose or create SBP invocations *i.e.,* the load in terms of the number of requests of all the copies of a given service is the same as the load of the original one without duplications/consolidations.

Property 2 The dynamics in terms of load evolution of the original process is preserved in the transformed one *i.e.,* for any reachable load distribution in the original net there is an equivalent (according to property 1) reachable load distribution in the transformed net.

We can also easily deduce that from properties 1 and 2 that duplication/consolidation properties preserve the call sessions dynamics.

5 Framework for the Evaluation of Elasticity Strategies

In order to manage the SBPs elasticity, several strategies can be used [11, 9, 20]. The strategy is responsible of making decisions on the execution of elasticity

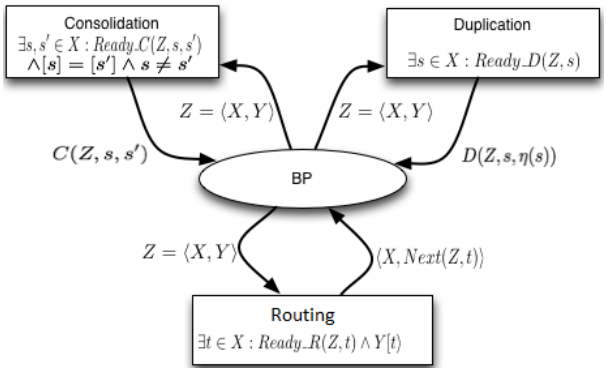

Fig. 3. High level Petri net (HLPN) of the generic controller

mechanisms *i.e.,* deciding when and how to use these mechanisms. So, it is necessary to ensure the precision of a strategy before using it to guarantee its effectiveness. The abundance of possible strategies requires their evaluation and validation. For this reason, we propose a framework, called generic controller, for the evaluation of SBPs elasticity strategies. This generic controller allows the implementation and execution of different elasticity strategies in order to analyze the behavior and the impact of these strategies on SBPs elasticity. Our controller has the capability to perform three actions:

- 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 resulted 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 removing of an unnecessary copy of a service in order to meet its workload decrease.

If we consider the three actions that can be performed by the elasticity controller, any combination of conditions associated with a decision of routing, duplication and consolidation is an elasticity strategy.

5.1 Formal Description of the Generic Controller

To model our generic controller we used high level Petri nets (HLPN). Due to the lack of space and the heaviness of notations of high level Petri nets, we give here, an informal definition; a more rigorous one can be found in [14]. As classic Petri nets, HLPN is a place-transition bipartite graph. The places are typed, a type can be any set of values (we denote by $type(p)$ the type of the place p). An arc

connecting a place p and a transition t is labeled by a multiset of expressions of type $type(p)$. Expression of a type $type(p)$ can be any values of $type(p)$, a variable or any function with domain $type(p)$. The transitions in HLPN can be guarded by a condition *i.e.,* expression of boolean type. The variables that appear in a transition condition and the expressions of its output arcs must be restricted to the variables that appear in the expressions of the input arcs. A marking of HLPN is any function that associates to each place p a multiset of $type(p)$. As in classical Petri nets, a HLPN system is composed of a HLPN and a marking. A transition is fireable, given a marking, iff there is a binding of the variables of its input arcs that validate the condition. The firing of a transition, given a binding, removes the instantiated multisets from input places and adds the instantiated multiset to the output places. Let us mention that the dynamics of an HLPN system can be obtained by computing the reachability graph exactly as classical Petri nets.

The structure of the controller is shown in Figure 3. The controller contains one place (BP) of type CPN system. The marking of this place is modified by the transitions of the controller after each firing:

- Routing: This transition is fireable if we can bind the variable Z to a CPN system $S = \langle N, M \rangle$ where there exists a transition t fireable in S and the predicate $Ready\_R(S, t)$ is satisfied. The firing of the Routing transition adds the CPN system S after the firing of t ($Next(Z, t)$ returns the marking after the firing of t).
- Duplication: This transition is fireable if we can bind the variable Z to a CPN system $S = \langle N, M \rangle$ where there exists a place s and the predicate $ready\_D(Z, s)$ is satisfied. The firing of the Duplication transition adds a new system resulted from the duplication of s in S.
- Consolidation: This transition is fireable if we can bind the variable Z to a CPN system $S = \langle N, M \rangle$ where there exists two copies of the same service, s and s', and the predicate $ready\_C(Z, s, s')$ is satisfied. The firing of the Consolidation transition adds a CPN system resulted from the consolidation of s' in S.

The elasticity conditions that decide when duplicate/consolidate a service are implemented in predicates $ready\_D$ (for duplication) and $ready\_C$ (for consolidation) while the condition that decides on how the service calls are routed is implemented in the predicate $ready\_R$. The execution of controller actions (Duplication/Consolidation and Routing) is performed after checking the guards of the execution of these actions ($ready\_D$, $ready\_C$, $ready\_R$). In our controller, the conditions are generic to allow the use of different elasticity strategies. By instantiating our generic controller, one can analyze and evaluate behaviors and performances of the implemented strategies.

5.2 How to Evaluate Elasticity Strategies with the Framework

The controller has been designed to offer developers a framework to define and evaluate elasticity strategies. In this section, we will show how a strategy developer can instantiate our controller to define elasticity strategies and evaluate

their behavior and performance. The execution of the instantiated controller generates the reachability graph of the controller which contains all the possible evolutions of the SBP with respect to the implemented strategy. As we will see, many properties can be checked and many indicators can be observed. The only restriction is to limit the number of calls during the analysis phase. Otherwise this would generate an infinite reachability graph. Note that there are tools to analyze unbounded HLPN nets but do not support any property. In this paper, we propose two kinds of evaluation:

Model Checking Evaluation Using a HLPN tool, the developer can generate the reachability graph of the controller which can then be analyzed using any model-checker and any temporal logic. Some significant examples of properties are given below:

- QoS violation: Let us assume that we associate for each service a maximal threshold over which its QoS will decrease drastically. Using temporal logic, one can check whether it is possible to reach a situation where one or some services have exceeded their thresholds *i.e.,* transfer a call to a copy of service that has already reached its maximal capacity.
- Blocked services: Let us suppose a routing strategy that allows only transition firing iff the next marking does not exceed the thresholds of some services. We can check if this strategy, coupled with a duplication strategy, would not cause a deadlock in the call transfer *i.e.,* there are fireable transitions in the SBP whereas the routing condition is no longer satisfied.
- Elasticity loop: Duplication and consolidation are costly activities. Given an elasticity strategy, one can check if this strategy can provoke a loop of elasticity *i.e.,* a duplication followed by consolidation of the same service while there is no (or few) calls arrival which means that the strategy causes unnecessary duplication of services.

Performance Evaluation The developer can also define a set of indicators to evaluate strategies' performance. For example an indicator that computes the number of copies of each service, etc. The value of these indicators will be calculated according to the evolution of the controller *i.e.,* each state of the reachability graph will contain the values of the indicators. The analysis of these indicators allows us to evaluate strategies' performance.

Many parameters can be evaluated, we will focus here on two parameters in order to answer two questions:

- How does the strategy influence the workload of the SBP according to the solicitations?
- How efficient is the resources allocation by the strategy to face the variation of the SBP solicitations?

We measure the workload of the SBP as the average of workloads of its basic services. To do so, we implemented an indicator which stores, at each step of

the SBP evolution, the average of the number of running instances on each of its basic services which can be obtained by dividing the number of tokens in the SBP net by the number of places. Concerning resources we consider the number of deployed services copies. We define two indicators. In the first indicator we store, at each step of the SBP evolution, the minimum number of each service copies needed to handle the current number of instances. Note that each copy of services can handle its maximum threshold instances. The second indicator will store the real number of the SBP services produced by a strategy.

5.3 Example of an Application of the Framework

We present hereafter an example, for a proof of concept, of strategies definition and evaluation with the framework. For that, we implemented the controller 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 [16].

Experimental Setup In order to illustrate the feasibility of our approach, we propose here to implement two elasticity strategies inspired from the literature [13, 5]. We applied such two strategies on the same SBP system $S = \langle N, M \rangle$ where N is the Petri net of an SBP composed by 3 services ($s1\_1, s2\_1, s3\_1$) executed in sequence and a data providing service sDB_1 for managing the state of the stateful service $s2\_1$. $M_0 = (0, 0, 0, 0)$ is its initial marking. An invocation (a call) of the SBP is represented by adding a token to a copy of the place $s1\_1$, the invocation takes end by removing a token from a copy of the place $s3\_1$.

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

- Max_t($s1\_1$) = 5. Max_t($s2\_1$) = 3. Max_t($s3\_1$) = 5. Max_t(sDB_1) = 5.
- Min_t($s1\_1$) = 1. Min_t($s2\_1$) = 1. Min_t($s3\_1$) = 1. Min_t(sDB_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.

Elasticity Strategies As we explained previously, the definition of a strategy consists in instantiating the three generic predicates *ready_R*, *ready_D* and *ready_C*. We use two threshold-based scaling algorithms that use the concept of maximum and minimum thresholds to make elasticity decisions. Note that initially these algorithms do not deal directly with the SPB elasticity but use a reasoning that can be used to manage the SPB elasticity. Here after the strategies:

Strategy 1 In [13] an algorithm is proposed to scale up or down an application instance by replication in response to a change in the workload.

- $Ready\_D(S, s) : M(s) \geqslant Max\_t(s) \wedge \nexists s' \in [s] : M(s') < Max\_t(s') \wedge \exists t \in^{\bullet} [s] : M[t\rangle$. It duplicates a copy s of service if all copies of this service have already reached theirs maximal threshold. In addition, there is a service call waiting to be transferred to this copy s.
- $Ready\_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min\_t(s) \wedge \nexists t \in^{\bullet} [s] : M[t\rangle$. 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 not service call waiting to be transferred to this copy s.
- $Ready\_R(S, t) : \forall s \in P : M'(s) < Max\_t(s)$ with $M[t\rangle M'$. It routes a call if this call transfer does not cause a violation of the maximum thresholds of services.

Strategy 2 In [5] a scaling algorithm is proposed to scale up or down the number of instances according to a threshold in each instance.

- $Ready\_D(S, s) : M(s) \geqslant Max\_t(s) \wedge \nexists s' \in [s] : M(s') < Max\_t(s')$. It duplicates a copy s of service if all copies of this service have already reached theirs maximal threshold.
- $Ready\_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min\_t(s)$. 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.
- $Ready\_R(S, t) : \forall s \in P : M'(s) < Max\_t(s)$ with $M[t\rangle M'$. Same routing strategy that strategy 1.

Strategy 3 To illustrate the elasticity impacts, we define also a third strategy that implements only a routing strategy.

- $Ready\_R(S, t) : \forall s \in P : M'(s) < Max\_t(s)$ with $M[t\rangle M'$. Same routing strategy that strategy 1 and strategy 2.

Evaluation of Strategies In our experiment, we used a Poisson process (with mean 2) to define a scenario of calls arrival on the SBP. This scenario was applied on the three strategies. For each strategy we generate, using the *SNAKES* tool, the reachability graph of the instantiated controller. This graph represents all the possible evolutions of the SBP in terms of routing, duplication and consolidation actions. Hereafter, we present the results of our experiment:

Analysis of Model Checking Evaluation The analysis of the reachability graph generated by the instantiated controller allows us to deduce some behavioral properties of the execution of the SBP controlled. These properties are

summarized in the table below:

	Strategy 1	Strategy 2	Strategy 3
Qos violation	No	No	No
Blocked services	No	No	Yes
Elasticity loop	No	Yes	-

The analysis of this table allows us to deduce some properties:

- All three strategies avoid QoS violations thanks to the routing strategy used by the three strategies.
- Unlike Strategy 3 that does not implement elasticity mechanisms, Strategies 1 and 2 avoid blocking states by duplicating overloaded services.
- We notice also a difference between the strategies 1 and 2 in the presence of a loop of elasticity. This difference is explained by the conditions of duplication used by both strategies. Indeed, the conditions of duplication used in strategy 1 are more difficult to verify than the conditions of the strategy 2. So, the controller using the strategy 2 will react faster to load increases. This fast reaction in some cases can cause unnecessary elasticity loops.

Analysis of Performance Evaluation The average evolution of resources consumption with strategies 1 and 2 on all possible executions of the SBP (about 6000 possible executions) is shown in Figure 4. The analysis of this figure shows that both strategies provide the elasticity of SBP by adapting its resources consumption according to the variation of resource demands which avoids resources oversizing. Also, the resources demand never exceeds the resources consumption. This guarantees the availability of resources to provide required QoS and avoid resources over-utilization.

The Figure 5-(a) represents the evolution of average workload of services on one possible execution of the controller. We notice a difference between the strategies in the reactivity to the requests variation. We can see that the strategy 2 is more reactive than the strategy 1. Indeed, the strategy 2 causes more duplication/consolidation than strategy 1. The evolution of resources consumption on one possible execution of the controller is shown in Figure 5-(b). We can see that both strategies adapt the resources consumption according to the resources demand. Using both strategies allows a better efficiency in resources consumption, but there is an under-utilization of resources in some periods.

The analysis of these figures shows a difference between the two strategies. This difference is explained by the conditions of elasticity used in these strategies. Indeed, the conditions of strategy 1 are more difficult to verify than the conditions of strategy 2 (the condition on the existence of service call waiting to be transferred). So, the controller using strategy 2 reacts faster. We can see that the reactivity of strategy 2 does not always mean better efficiency. In fact, this reactivity can cause unnecessary duplication of services.

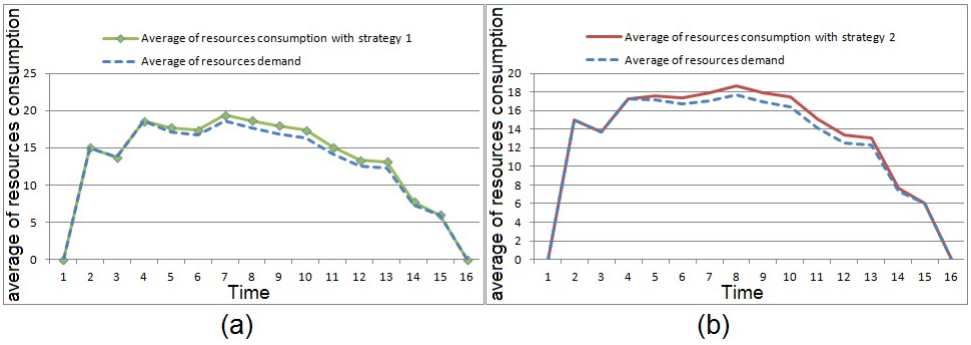

Fig. 4. The average evolution of resources consumption (a) with strategy 1 (b) with strategy 2

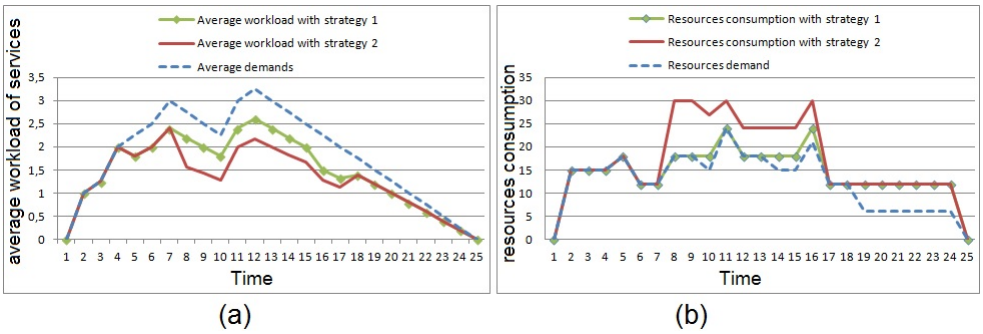

Fig. 5. (a) The evolution of average workload of services (b) The evolution of resources consumption

6 Conclusion

This paper addresses the problem of elasticity of stateful SBPs deployed in Cloud environments. Unlike existing work, our approach tackles the elasticity at the level of SBPs. To perform stateful SBPs elasticity we proposed and formalized using colored Petri nets two operations: Duplication and consolidation. In addition, we have proposed a framework to define elasticity strategies and two approaches to evaluate elasticity strategies. Moreoever, we presented an example for the proof of concept. As perspectives of this work, we are working on the integration of the temporal aspect in our model. We also consider the implementation of the elasticity operations into *CloudServ* (a PaaS under development).

References

1. M. Amziani, T. Melliti, and S. Tata. A generic framework for service-based business process elasticity in the cloud. In *BPM*, 2012.
2. M. Amziani, T. Melliti, and S. Tata. Formal modeling and evaluation of service-based business process elasticity in the cloud. In *WETICE*, 2013.
3. J. Bi, Z. Zhu, R. Tian, and Q. Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. *IEEE CLOUD*, 2010.
4. R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Gener. Comput. Syst.*, 2012.
5. T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *ICEBE*, 2009.
6. T. N. B. Duong, X. Li, and R. S. M. Goh. A framework for dynamic resource provisioning and adaptation in iaas clouds. *CloudCom*, 2011.
7. S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. *IEEE Internet Computing*, 2011.
8. S. Dutta, S. Gera, A. Verma, and B. Viswanathan. Smartscale: Automatic application scaling in enterprise clouds. In *IEEE CLOUD*, 2012.
9. G. Galante and L. de Bona. A survey on cloud computing elasticity. In *IEEE International Conference on Utility and Cloud Computing (UCC)*, 2012.
10. J. Geelan, M. Klems, R. Cohen, J. Kaplan, D. Gourlay, P. Gaw, D. Edwards, B. de Haaff, B. Kepes, K. Sheynkman, O. Sultan, K. Hartig, J. Pritzker, T. Doerksen, T. von Eicken, P. Wallis, M. Sheehan, D. Dodge, A. Ricadela, B. Martin, B. Kepes, and I. W. Berger. Twenty-One Experts Define Cloud Computing.
11. H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Exploring alternative approaches to implement an elasticity policy. In *IEEE CLOUD*, 2011.
12. R. Han, L. Guo, Y. Guo, and S. He. A deployment platform for dynamically scaling applications in the cloud. *CloudCom*, 2011.
13. S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han. Elastic application container: A lightweight approach for cloud resource provisioning. *AINA*, 2012.
14. K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*. Springer, USA, 1997.
15. P. Kranas, V. Anagnostopoulos, A. Menychtas, and T. A. Varvarigou. ElaaS: An Innovative Elasticity as a Service Framework for Dynamic Management across the Cloud Stack Layers. In *CISIS*, 2012.
16. F. Pommereau. Nets in nets with snakes. In *Int. Workshop on Modelling of Objects, Components, and Agents*, 2009.
17. B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *International Joint Conference on INC, IMS and IDC*, NCM, 2009.
18. N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. *IEEE CLOUD*, 2011.
19. W.-T. Tsai, X. Sun, Q. Shao, and G. Qi. Two-tier multi-tenancy scaling and load balancing. In *ICEBE*, 2010.
20. L. M. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 2011.
21. S. Yangui, M. Mohamed, S. Tata, and S. Moalla. Scalable service containers. In *CloudCom*, 2011.