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

Mourad Amziani*†, Tarek Melliti†, Samir Tata*
*Institut Mines-Telecom, TELECOM SudParis, UMR CNRS Samovar, Evry, France
†IBISC, University of Evry Val d'Essonne, Evry, France

***Abstract*—Cloud computing is a new delivery model for IT services. Cloud platforms are being increasingly used for the deployment and execution of service-based business processes (SBPs). Nevertheless, the provisioning of elastic infrastructures and/or platforms is not sufficient to provide users with elasticity at the level of SBPs. Therefore, there is a need to provide SBPs with mechanisms to scale their resource requirements up and down whenever possible. This can be achieved using mechanisms for duplicating and consolidating business services that compose the SBPs. In this paper, we propose a formal model for SBPs elasticity in the Cloud. We show that our model preserves the semantics of SBPs when services are duplicated or consolidated. We also propose a formal framework for the evaluation of elasticity strategies that decide on when and how many resources are required to ensure elasticity of SBPs.**

I. Introduction

Cloud computing is a new delivery model for IT services based on Internet protocols. It typically involves provisioning of dynamically scalable and often virtualized resources at the infrastructure, platform and software levels. Cloud environments are being increasingly used for deploying and executing business processes and particularly service-based business processes (SBPs) that are made up of components that provide business services. One of the expected facilities of Cloud environments is elasticity at different levels.

At the platform as a service (PaaS) level, the deployed processes should be provided with platform mechanisms that can scale up and down whenever needed. In this context, we have conducted studies of existing application servers and SBP engines. These classical platforms are not elastic [10]. For that reason, we have developed a new model for service deployment called micro-container [15]. Our approach was based on a simple idea that consists in dedicating a micro-container to each deployed service in Cloud environment. Each micro-container can host and run its service and can be then seen as a specialized container. Generated micro-container provides minimal and personalized functionalities to manage the life cycle of the deployed services. With this idea we have shown the elasticity of Cloud at the PaaS level can be ensured. We have actually conducted conclusive experiments on classical service containers (like Apache Axis2) to validate the elasticity of micro-containers [15]. In addition, we have shown that micro-containers can be used to host service-based application and particularly SBPs. Nonetheless, provisioning of elastic platforms, *e.g.,* based on micro-containers, is not sufficient to provide elasticity of the deployed business process (at the Software as a Service (SaaS) level). 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 evaluate elasticity strategies of SBPs?

Performing elasticity consists in 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.

Many strategies that decide on when SBP elasticity is performed can be proposed. Some strategies use minimal and maximal threshold of load in each business service to make elasticity decisions. Some others use predictive techniques. It would be useful for a Cloud provider to have an evaluation framework in order to make a better decision on the elasticity strategy to adopt.

In this paper, we propose a formal model for SBP elasticity and a framework to evaluate elasticity strategies.

The rest of this paper is organized as follows. In section II, which is dedicated to the first question we raised above, we propose a formal model for elasticity of SBPs. In this model, processes are defined as Petri nets. Then, elasticity operations (duplication and consolidation) are defined and their correctness is proven. In section III, which is dedicated to the second question we raised above, we propose a controller that provides a framework for the evaluation of elasticity strategies. We present how to evaluate elasticity strategies with the proposed controller. An example, for a proof of concept, is also detailed. Section IV presents related work and Section V concludes the paper.

II. Formal Model for SBPs Elasticity

A SBP is a business process that consists in assembling a set of elementary IT-enabled services. These services realize the business activities of the considered SBP. Assembling

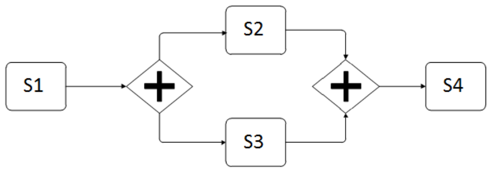

Figure 1. SBP of the example of the online computer shopping service

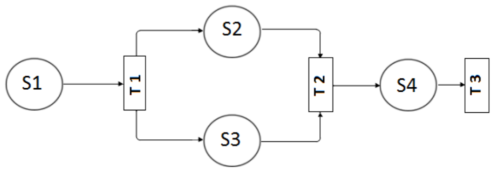

Figure 2. Petri net of the SBP of the online computer shopping service

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

We present in the following a motivating example for our approach (Section II-A) and then, a formal model to describe the load of SBPs (Section II-B) and two elasticity operations (Section II-C) and we prove their correctness (Section II-D).

A. Motivating Example

The example of Figure 1 presents a SBP for an online computer shopping service composed of four services and modeled in BPMN:

- Service requests ($S1$): receives requests to purchase a computer (processing time = 1s).
- Service components assembly ($S2$): performs the assembly of computer components according to the desired characteristics (processing time = 60s).
- Service invoice ($S3$): creates the invoice related to the purchased computer (processing time = 1s).
- Service delivery ($S4$): delivers the computer with its invoice to the customer (processing time = 1s).

We suppose that the SBP receives 100 requests. The response time of the SBP will be equal to: 62s (processing time of a request) x 100 (number of requests) = 6200s.

One solution to improve the response time would be to duplicate the SBP to support the load. So to support 100 requests; we will have 100 copies of the SBP in parallel (one request by SBP). The response time for processing all requests will be equal to: 62s (the 100 treatments are done in parallel). This solution solves the problem of SBP response time, but it creates a constraint on the consumption of resources. Indeed, the disadvantage of this solution is that it duplicates all the services of the SBP.

We think that it is not necessary to duplicate all the services of the SBP while the bottleneck comes from a single service of the SBP (or a certain number of services). We think that the duplication of this service will solve the response time problem while avoiding unnecessary resources consummation. So, back to our example, compared to $S1$, $S3$ and $S4$, $S2$ is a time and resources consuming service, it is obvious that when considering elasticity, duplicating or consolidating instances of $S2$ could be enough. So, we will have 100 copies of S2 in parallel (but only one copy of the services S1, S3 and S4). In this case, the response time for processing these requests will be equal to 62s.

B. SBP Modeling

To model SBPs, several techniques can be used. In our work, we are interested in the formal aspect of the model. So, we model the SBP using Petri nets. Many approaches model SBPs using Petri nets, but instead of focusing on the execution model of processes and their services, we focus on the dynamic (evolution) of loads on each basic service of the SBP's composition. In our model, each service is represented by at least one place (the set of places of each service are related with an equivalence relation). The transitions represent calls transfers between services according to the behavioral specification of the SBP. The modeling of the SBP of Figure 1 gives the Petri net shown in Figure 2.

Definition 1: A SBP load model is a Petri net $N =< P, T, Pre, Post, \equiv_P, \equiv_T >$:

- P: a set of places (represents the set of services/activities involving in a SBP).
- T: a set of transitions (represents the call transfers between services in a SBP).
- $Pre : P \times T \rightarrow \{0, 1\}$
- $Post : T \times P \rightarrow \{0, 1\}$
- $\equiv_P \subseteq P \times P$: an equivalence relation over P
- $\equiv_T \subseteq T \times T$: an equivalence relation over T.

For a place p and a transition t we give the following notations:

- $[p]_{\equiv_P} = \{p' | (p, p') \in \equiv_P\}$. $[t]_{\equiv_T} = \{t' | (t, t') \in \equiv_T\}$
- $p^\bullet = \{t \in T | Pre(p, t) = 1\}$
- ${}^\bullet p = \{t \in T | Post(t, p) = 1\}$
- $t^\bullet = \{p \in P | Post(t, p) = 1\}$
- ${}^\bullet t = \{p \in P | Pre(p, t) = 1\}$

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 2: Let N be a Petri net, we define a net system $S = \langle N, M\rangle$ with $M : P \rightarrow \mathbb{N}$ a marking that associates to each place an amount of tokens. The marking is also extended to equivalent classes *i.e.,* $M([p]) = \sum\limits_{p' \in [p]} M(p')$. The marking of a Petri net represents a distribution of calls over the set of services that compose the SBP. A Petri net system models a particular distribution of calls over the services of a deployed SBP.

Definition 3: Given a net system $S = \langle N, M\rangle$ we say that a transition t is fireable in the marking M, noted by $M[t\rangle$ iff $\forall p \in^{\bullet} t : M(p) \geq 1$. A class of transitions is fireable in M, $M[[t]\rangle$, iff $\exists t' \in [t] : M[t'\rangle$

Definition 4: The firing of a transition t in marking M changes the marking to M' s.t. $\forall p : M'(p) = M(p) + (Post(t,p) - Pre(p,t))$. We note the transition by $M[t\rangle M'$. We extend the transition notation to classes using $M[[t]\rangle M'$ where $M' \in \{M'' | \exists t' \in [t] : M[t'\rangle M''\}$.

C. Elasticity Operations

Place Duplication: When a service has a lot of calls, it will be overloaded and this can lead to loss of QoS. A solution to this overflow problem is to duplicate the service without changing underlying SBP. Hereafter, we give the definition of an operator that duplicates a service.

Definition 5: Let $S = \langle N, M\rangle$ be a net 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 net system $S' = \langle N', M'\rangle$ s.t

- $P' = P \cup \{p^c\}$
- $T' = T \cup T''$ 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).
- $Pre' : P' \times T' \rightarrow \{0, 1\}$
- $Post' : T' \times P' \rightarrow \{0, 1\}$
- $\equiv_{P'} \subseteq P' \times P'$ 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.
- $M' : P' \rightarrow \mathbb{N}$ with $M'(p') = M(p')$ if $p' \neq p^c$ and 0 otherwise.

The Pre' (respectively $Post'$) functions are defined as follow:

$$Pre'(p', t') = \begin{cases} Pre(p', t') & p' \in P \wedge t' \in T \\ Pre(p', t) & t \in T \wedge t' \in (T' \setminus T) \\ & \wedge t' \in [t]_{\equiv_{T'}} \\ & \wedge p' \in (P \setminus \{p\}) \\ Pre(p, t) & t \in T \wedge t' \in (T' \setminus T) \\ & \wedge t' \in [t]_{\equiv_{T'}} \wedge p' = p^c \\ 0 & otherwise. \end{cases}$$

$Post'$ can be obtained by replacing Pre by $Post$.

Place Consolidation: When a service has few calls, the containers that host its instances use more resources than required. As a dual operator to duplication we define the consolidation operator that removes a copy of a service.

Definition 6: Let $S = \langle N, M\rangle$ be a net 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 net system $S' = \langle N', M'\rangle$ s.t

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

We call well-defined net any net where the equivalent relation over places and transitions are composed of copies resulted from duplication and/or consolidation operators.

Definition 7 (Well-defined Net): Let N_0 be a net where $\equiv_{P_0}$ and $\equiv_{T_0}$ are the identity relations. We call here well-defined net, any net N resulted from a finite application of duplication and/or consolidation operators on N_0.

Proposition 1: Let N be a well-defined net, the following properties are held on N:

$$\forall t_1, t_2 \in T : [t_1] = [t_2] \Rightarrow [{}^{\bullet}t_1] = [{}^{\bullet}t_2] \wedge [t_1^{\bullet}] = [t_2^{\bullet}] \quad (1)$$

$$\begin{aligned} \forall p_1, p_2 \in P, t \in T : p_1, p_2 \in ({}^{\bullet}t \cup t^{\bullet}) \\ \Rightarrow p_1 = p_2 \vee [p_1] \cap [p_2] = \emptyset \end{aligned} \quad (2)$$

$$\forall t \in T : |[t]| = \prod_{p \in ({}^{\bullet}t \cup t^{\bullet})} |[p]| \quad (3)$$

Proof: The proof of equations 1 and 2 can be derived from the definition of the duplication and consolidation operators. We prove equation (3) by recurrence. Consider a well-defined net system $S = \langle N, M\rangle$. If $\equiv_T$ and $\equiv_P$ are the identity relations, then we have the equation 3 holds for N this because of $|[x]| = 1$ for any $x \in P \cup T$.
Consider now $\equiv_T$ and $\equiv_P$ are different from the identity relations in S and suppose that equation 3 is true for S *i.e.,*

$$|[t]_{\equiv_T}| = \prod_{p \in ({}^{\bullet}t \cup t^{\bullet})} |[p]_{\equiv_P}|$$

By equations (1) and (2) we deduce for some $p \in ({}^{\bullet}t \cup t^{\bullet})$ (w.l.o.g let we take $p \in^{\bullet} t$) that:

$$|p^{\bullet} \cap [t]_{\equiv_T}| = \frac{|[t]_{\equiv_T}|}{[p]_{\equiv_P}} = \prod_{p' \in ({}^{\bullet}t \cup t^{\bullet}) \wedge p' \neq p} |[p']_{\equiv_P}|.$$

Let we create a net system S' by duplicating p. Then by definition of duplication we will add $|p^{\bullet} \cap [t]_{\equiv_T}|$ of copies of t, So:

$$\begin{gathered} |[t]_{\equiv_{T'}}| = |[t]_{\equiv_T}| + |p^{\bullet} \cap [t]_{\equiv_T}| = \\ \left(\prod_{p \in ({}^{\bullet}t \cup t^{\bullet})} |[p]_{\equiv_P}|\right) + \left(\prod_{p' \in ({}^{\bullet}t \cup t^{\bullet}) \wedge p' \neq p} |[p']_{\equiv_P}|\right) = \\ \left(\prod_{p' \in ({}^{\bullet}t \cup t^{\bullet}) \wedge p' \neq p} |[p']_{\equiv_P}|\right) * (1 + |[p']_{\equiv_P}|) = \\ \left(\prod_{p' \in ({}^{\bullet}t \cup t^{\bullet}) \wedge p' \neq p} |[p']_{\equiv_{P'}}|\right) * (|[p']_{\equiv_{P'}}|) = \\ \prod_{p' \in ({}^{\bullet}t \cup t^{\bullet})} |[p']_{\equiv_{P'}}| \end{gathered}$$

■

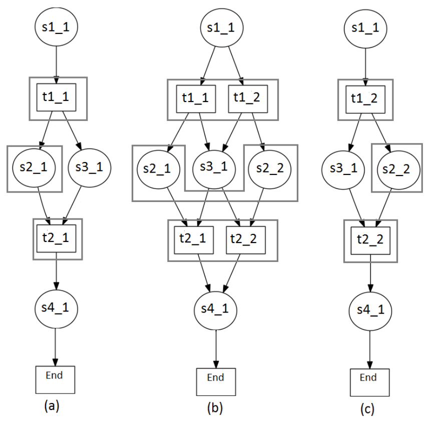

Figure 3. Example of the elasticity of a SBP

The proof of the equation (3) for consolidation can be done in the same manner.

Example 1: Figure 3-(a) shows an example of nets system that represents the SBP *computer shopping* described previously. The relations $\equiv_P$ and $\equiv_T$ are the identity relations. Figure 3-(b) is the resulted system from the duplication of $s2\_1$. Figure 3(c) is the consolidation of the place $s2\_1$ in its copy $s2\_2$. The boxes represent the equivalence relations.

D. Correctness of Elasticity Operations

In order to guarantee that the semantics of the SBP is preserved by duplication and consolidation operators we must prove that according to some equivalence relation, any sequence of transformation on a SBP is equivalent, according to some properties, to the original one. In our case here, as mentioned previously, the Petri net of a SBP does not denote an execution model but a dynamic view on the evolution of the SBP load (the marking). Therefore, we want to keep the same view of load evolution regardless of any transformation of a net. In order to do so, the following two properties have to be preserved:

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 must be 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 must be 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 give now a definition of an equivalence relation between net systems that cover the two previous properties.

Definition 8 (Equivalence relation): Let $S = \langle N, M\rangle$ and $S' = \langle N', M'\rangle$ be two net systems. Let $\rho_1 : [P]_{\equiv_P} \rightarrow [P']_{\equiv_{P'}}$ (resp. $\rho_2 : [T]_{\equiv_T} \rightarrow [T']_{\equiv_{T'}}$) be two bijective functions that associates to each equivalent class in P (resp. in T) an equivalent class in P' (resp. in T'). We use ρ as the union of the two functions. Two net systems S and S' are equivalent according to ρ, noted by $S \sim_\rho S'$, iff:

(a) $\forall p \in P : M([p]_{\equiv_P}) = M'(\rho([p]_{\equiv_P}))$

(b) $\forall t \in T : M[t\rangle M_1 \Rightarrow \exists t' \in \rho([t]_{\equiv_T}) : M'[t'\rangle M'_1 \wedge \langle N, M_1\rangle \sim_\rho \langle N', M'_1\rangle$

(c) $\forall t' \in T' : M'[t'\rangle M'_1 \Rightarrow \exists t \in \rho^{-1}([t']_{\equiv_{T'}}) : M[t\rangle M_1 \wedge \langle N, M_1\rangle \sim_\rho \langle N', M'_1\rangle$

Proposition 2: Let N be a well-defined net and t a transition. If we consider $\{p_1, ..., p_n\} = ^\bullet t \cup t^\bullet$ then we have: $\forall p'_1, ..., p'_n \subseteq P : \bigwedge_{i=1...n} p'_i \in [p_i] \Rightarrow \exists t' \in [t] : \{p'_1, ..., p'_n\} = ^\bullet t' \cup t'^\bullet$

Proof: The proposition states that any combination of places taken from the equivalence classes of the pre-places of a transition is also a set of pre-places of one of its copies (resp. post-places). While by definition we can show that for any transitions t, t' s.t. $(t, t') \in_{\equiv_T}$ we have $|^\bullet t| = |^\bullet t'|$ and $|t^\bullet| = |t'^\bullet|$ by the constraints of the equations (1) to (3) we can conclude the proof of the proposition. ■

Proposition 3: Let $S = \langle N, M\rangle$ and $S' = \langle N', M'\rangle = D(S, p_1, p_1^c)$. Let ρ defined as the union of $\rho_1([p]_{\equiv_P}) = [p]_{\equiv_{P'}}$ for any $p \in P$ and $\rho_2([t]_{\equiv_T}) = [t]_{\equiv_{T'}}$ for any $t \in T$. Let now some M_1 ,M'_1 two reachable markings, respectively from M and M', such that $\forall p \in P : M_1([p]_{\equiv_P}) = M'_1(\rho([p]_{\equiv_P}))$ then we have:
$\exists t \in T : M_1[t\rangle \Leftrightarrow \exists t' \in \rho([t]_{\equiv_T}) : M'_1[t'\rangle$.

Proof: let t be a fireable transition from M_1:
$(\Leftrightarrow) \forall p \in ^\bullet t : M_1(p) \geq 1$
$(\Leftrightarrow) \forall p \in ^\bullet t : M_1([p]_{\equiv_P}) \geq 1$
$(\Leftrightarrow) \forall p \in ^\bullet t : M'_1(\rho([q]_{\equiv_P})) \geq 1$
$(\Leftrightarrow) \forall p \in ^\bullet t : \exists p' \in \rho([p]_{\equiv_P}) : M'_1(p') \geq 1$
according to proposition 2
$(\Leftrightarrow) \exists t' \in [t]_{\equiv_{T'}} : M'_1[t'\rangle$
$(\Leftrightarrow) \exists t' \in \rho([t]_{\equiv_T}) : M'_1[t'\rangle$ ■

Theorem 1: Let $S = \langle N, M\rangle$ with N a well-defined nets and let $S' = \langle N', M'\rangle = D(S, p_1, p_1^c)$ we have:
$S \sim_\rho S'$ where ρ is defined as in proposition 3.

Proof: First, by definition, ρ is a bijection.

(a) By definition of duplication, we have $M'(p_1^c) = 0$:
$$\forall p \in P : M([p]_{\equiv_P}) = M'(\rho([p]_{\equiv_P})) \quad (4)$$

(b) We show now that for any two markings M_1 and M'_1 respectively reachable from M and M' s.t $\forall p \in P : M_1([p]_{\equiv_P}) = M'_1(\rho([p]_{\equiv_P}))$ then for all transitions fireable from M_1 we can fire one of its copies in S' from M'_1 and reach two markings that conserve the marking over equivalent classes under the same ρ (and vice versa).
Let $M_1[t_1\rangle M_2$, according to proposition 3 we have $M'_1[t'_1\rangle M'_2$ with $t'_1 \in \rho([t_1]_{\equiv_T})(t'_1 \in [t_1]_{\equiv_{T'}})$.
From equation 2 we know that only one place in an equivalent class will be concerned by the firing of t_1 (idem for t'_1) so:

$$\begin{gathered}\forall p \in P : M_2([p]_{\equiv_P}) = \\ M_1([p]_{\equiv_P}) + (Post(t,p) - Pre(p,t)) \wedge \\ \forall p' \in P' : M_2'([p']_{\equiv_{P'}}) = \\ M_1([p']_{\equiv_{P'}}) + (Post'(t',p') - Pre'(p',t')) \end{gathered} \quad (5)$$

We know also, by definition of the duplication:

$$\begin{gathered}\forall p \in P, p' \in P', t \in T, t' \in T' : [p]_{\equiv_P} \subseteq \\ [p']_{\equiv_{P'}} \wedge [t]_{\equiv_T} \subseteq [t']_{\equiv_{T'}} \Rightarrow \\ Pre(p,t) = Pre'(p',t') \wedge Post(t,p) = Post'(t',p') \end{gathered} \quad (6)$$

From equations (5) and (6) we conclude:

$$\forall p \in P : M_2([p]_{\equiv_P}) = M_2'([p]_{\equiv_{P'}}) \quad (7)$$

and so $\forall p \in P : M_2([p]_{\equiv_P}) = M_2'(\rho([p]_{\equiv_P}))$

(c) The proof of point (c) is similar to (b).

We can conclude that:

$$\langle N, M_2 \rangle \sim_\rho \langle N', M_2' \rangle \quad (8)$$

and By induction on equations (4) and (8) we conclude that $S \sim_\rho S'$ ■

Theorem 2: Let $S = \langle N, M \rangle$ with N a well-defined net and let $S' = \langle N', M' \rangle = C(S, p, p^c)$ we have:

$S \sim_\rho S'$ where ρ is defined as in proposition 3.

Proof: For any a well-defined net system S we can always construct a net system S_0 where S is resulted from a sequence of duplication on S_0. By theorem 1 $S \sim_\rho S_0$. If we consolidate a place in S producing the net system S' then S' is also a result from a sequence of duplication on S_0 and so $S' \sim_\rho S_0$. We conclude that $S \sim_\rho S'$ ■

By theorem 1 and 2 we proved that for any finite sequence of application duplication and/or consolidation operators on a well-defined net is also a well-defined net that preserves the semantic of the original net.

III. A Framework for the Evaluation of Elasticity Strategies

Usually, a set of policies is implemented in what is usually called controller to guarantee some SLA properties. In our case, we are interested in elasticity policies of services that compose a SBP. In order to achieve this, we want to develop a controller to provide an optimal ratio QoS and allocated resources of a SBP. Our controller will have 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.

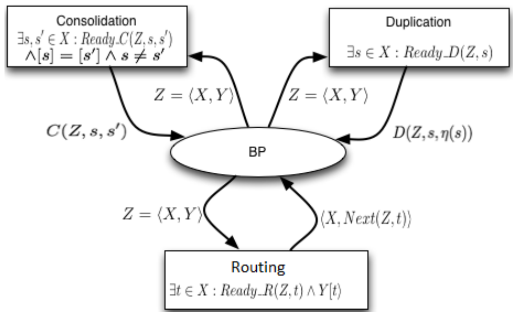
Consolidation
$\exists s, s' \in X : Ready\_C(Z,s,s') \wedge [s] = [s'] \wedge s \neq s'$
Duplication
$\exists s \in X : Ready\_D(Z,s)$
$Z = \langle X, Y \rangle$
$Z = \langle X, Y \rangle$
$C(Z,s,s')$
$D(Z,s,\eta(s))$
BP
$Z = \langle X, Y \rangle$
$\langle X, Next(Z,t) \rangle$
Routing
$\exists t \in X : Ready\_R(Z,t) \wedge Y[t]$

Figure 4. General architecture of the generic controller

If we consider the three actions that can be performed by an elasticity controller, any combination of conditions associated with a decision of routing, duplication and consolidation is an elasticity strategy. The strategy is responsible of making decisions on the execution of elasticity 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 the effectiveness of the duplication/consolidation mechanisms.

Several strategies can be used to manage the SBP elasticity [6]. The abundance of possible strategies requires their evaluation. For this, we propose a framework which is a generic controller that allows testing, evaluation and validation of different elasticity strategies.

A. Formal Description of the Generic Controller

As pointed out in the introduction our goal here is not to propose an additional elasticity strategy, but a framework, called generic controller, to evaluate different strategies. In order to allow analysis and evaluation of instantiated controller, we propose to model the controller as a high level Petri net (HLPN). Indeed, one can by instantiating our generic controller evaluate the performance of the implemented strategies using the HLPN analysis tools.

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 [9]. 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.

B. HLPN of the Generic Controller

The structure of the controller is shown in Figure 4-(b). The controller contains one place (BP) of type net system. The marking of this place is modified by the transitions of the controller after each firing. As mentioned before, the controller has three transitions:

- Routing: This transition is fireable if we can bind the variable Z to a net 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 net 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 net system $S = \langle N, M \rangle$ where there exists a place s and the predicate $ready\_D$ 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 net system $S = \langle N, M \rangle$ where there exists two copies of the same service, s and s', and the predicate $ready\_C$ is satisfied. The firing of the Consolidation transition adds a net system resulted from the consolidation of s' in S.

The execution of these actions is performed after checking the guards of the execution of these actions (ready_R, ready_D, ready_C). In our controller, the conditions are generic to allow the use of different elasticity strategies.

C. How to Evaluate Elasticity Strategies with the Controller

In this section, we will show how a strategy developer can use our controller to evaluate an elasticity strategy. The first step starts by instantiating the conditions of each controller transition *i.e.,* defining a strategy for routing, duplicating and consolidating. Then the SBP to be controlled along with the strategy should be defined. Then, the developer defines the calls arrival scenario to specify the way in which calls arrive to the SPB. Note here that it is possible to define the calls arrival using a Poisson process or by adding a new transition in the controller that simulates the arrival of calls and can change at any time the marking of the net system. Using a HLPN tool, the developer can then generate the reachability graph of the controller. The reachability graph contains all the possible evolutions of the SBP with respect to the implemented strategies. The resulted graph can then be analyzed using any model-checker. 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.
- 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 BP net system 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.

As we can see, many properties can be checked and many indicators can be observed on an instantiated controller. 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.

D. Example of an Application of the Framework

We present hereafter an example, for a proof of concept, of strategies evaluation with the controller. 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 [11].

1) Experimental Setup: In order to illustrate the feasibility of our approach, we propose here to implement two elasticity strategies inspired from the literature [8], [3]. We applied such two strategies on the same SBP system $S = \langle N, M \rangle$ where N is the net of Figure 3-(a) and M0=(0,0,0,0) 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 $s4\_1$.

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($s1\_1$) = 12. Max_t($s2\_1$) = 3. Max_t($s3\_1$) = 15. Max_t($s4\_1$) = 12.
- Min_t($s1\_1$) = 2. Min_t($s2\_1$) = 1. Min_t($s3\_1$) = 3. Min_t($s4\_1$) = 5.

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.

2) 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:

Strategy 1: In [8] a scaling algorithm is proposed to scale up or down an application instance in containers in response to a change in the application instance usage. The algorithm scales up by replicating an application instance, and scales down by removing a replicated application instance.

- $Ready\_D(S, s) : M(s) \geqslant Max\_t(s) \wedge \nexists s' \in [s] : M(s') < Max\_t(s') \wedge \exists t \in^{\bullet} [s] : M[t\rangle$
- $Ready\_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min\_t(s) \wedge \nexists t \in^{\bullet} [s] : M[t\rangle$
- $Ready\_R(S, t) : \forall s \in P : M'(s) < Max\_t(s)$ with $M[t\rangle M'$

Strategy 2: In [3] a scaling algorithm is proposed for automated provisioning of resources. The algorithm scales automatically the number of web servers according to a threshold in each web server instance.

- $Ready\_D(S, s) : M(s) \geqslant Max\_t(s) \wedge \nexists s' \in [s] : M(s') < Max\_t(s')$
- $Ready\_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min\_t(s)$
- $Ready\_R(S, t) : \forall s \in P : M'(s) < Max\_t(s)$ with $M[t\rangle M'$

3) Evaluation of Strategies: In our experiment, we used an invocation scenario that represents a calls arrival on the SBP. This scenario was applied on both strategies 1 and 2. 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. The analysis of this graph allows us to evaluate and compare the strategies. As claimed before many properties can be studied, we will focus here on two properties in order to answer two questions:

- How does the strategy influence the workload of the SBP face to the SBP 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. This indicator 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.

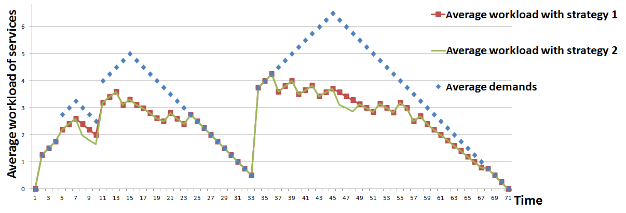

Figure 5. The evolution of average workload of services

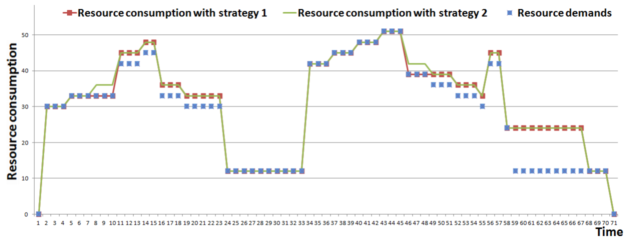

Figure 6. The evolution of resource consumption

The Figure 5 represents the evolution of average workload of services. The evolution of resource consumptions is shown in Figure 6.

4) Analysis of Results: The analysis of these figures allows us to deduce some properties: In Figure 5, we can see that both strategies adapt to the requests variation by using elasticity mechanisms. This shows that these two strategies guarantee the elasticity of the SBP. We notice also 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. This difference is explained by the conditions of elasticity used in both strategies. Indeed, the conditions of strategy 1 are more difficult to verify than the conditions of strategy 2. So, the controller using strategy 2 reacts faster. In Figure 6, we can see that both strategies adapt the resources consumption according to the resources demand 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 over-utilization. The use of 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 that the reactivity of strategy 2 does not always mean better efficiency in resources consumption than strategy 2. The example of Figure 6 shows that this reactivity can cause unnecessary consumption of resources. This is explained by the conditions of elasticity used in strategy 2 which can cause unnecessary duplication of services.

IV. Related Work

The elasticity in the Cloud at the IaaS level has been extensively studied in the past. Proposed approaches use generally sets of rules to make decisions about the elasticity of the infrastructure to adapt the amount of resources allocated according to user requests. In this kind of approaches, several techniques based on analytical or stochastic methods have been used. In [4], [12] the authors propose an approach which consists in adding or removing resources to virtual machines (VMs) to prevent over-loading and under-loading. In [7], [2] the authors propose to calculate the optimal number of VMs to be deployed according to demands variations. These approaches allow the elasticity of VMs but they are not sufficient to ensure the elasticity of the deployed applications since they do not take into account the nature of the application. In fact, each application has a maximal capacity, beyond this capacity the QoS decreases and can lead to the stuck of the container and by the same way the crash of the application. Giving to the container more physical resources will not solve the problem [15].

In [5] 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.

Duplication/consolidation mechanisms have been considered in the area of dynamic service deployment [14]. The proposed mechanisms allow the duplication/consolidation of the entire SBP and so, of all its services while the bottleneck may come from some services. In [13], 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 [1] we considered the elasticity of SBPs and provided duplication and consolidation mechanisms. In this work we go further in formally proving the correctness of our elasticity mechanisms. In addition, we provide a framework to evaluate duplication/consolidation based strategies.

At the best of our knowledge, the approaches for elasticity mainly those we cite above, are interested in the IaaS level. As stated before, ensuring elasticity at the IaaS level is not sufficient. Similarly, ensuring elasticity at the PaaS level is not enough to ensure elasticity of deployed SBPs. We believe that elasticity should be tuned at different levels of Cloud environments. We have already contributed to the elasticity at the PaaS level [15]. The work we present in this paper is novel in the sense that it (1) tackles the problem of elasticity at the SaaS level and (2) is based on a formal model and (3) proposes a framework for evaluating elasticity strategies.

V. Conclusion

This paper addresses the problem of elasticity of SBPs deployed in Cloud environments. Unlike existing work, our approach tackles the elasticity at the level of SBPs. To perform elasticity we proposed and formalized using Petri nets two operations: Duplication and consolidation. We showed that our formal model preserves the semantics of SBP. In addition, we have proposed a controller to evaluate elasticity strategies and given an example for the proof of concept. As perspectives of this work, we are working on the elasticity of stateful SBPs.

References

[1] M. Amziani, T. Melliti, and S. Tata. A generic framework for service-based business process elasticity in the cloud. *BPM*, 2012.

[2] J. Bi, Z. Zhu, R. Tian, and Q. Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. *IEEE CLOUD*, 2010.

[3] 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.

[4] T. N. B. Duong, X. Li, and R. S. M. Goh. A framework for dynamic resource provisioning and adaptation in iaas clouds. *CloudCom*, 2011.

[5] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. *IEEE Internet Computing*, 2011.

[6] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Exploring alternative approaches to implement an elasticity policy. In *IEEE CLOUD*, 2011.

[7] R. Han, L. Guo, Y. Guo, and S. He. A deployment platform for dynamically scaling applications in the cloud. *CloudCom*, 2011.

[8] 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.

[9] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*. Springer, USA, 1997.

[10] M. Mohamed, S. Yangui, S. Moalla, and S. Tata. Web service micro-container for service-based applications in cloud environments. In *WETICE*, 2011.

[11] F. Pommereau. Nets in nets with snakes. In *Int. Workshop on Modelling of Objects, Components, and Agents*, 2009.

[12] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. *IEEE CLOUD*, 2011.

[13] W.-T. Tsai, X. Sun, Q. Shao, and G. Qi. Two-tier multi-tenancy scaling and load balancing. In *ICEBE*, 2010.

[14] J. B. Weissman, S. Kim, and D. England. A framework for dynamic service adaptation in the grid: Next generation software program progress report. *IPDPS*, 2005.

[15] S. Yangui, M. Mohamed, S. Tata, and S. Moalla. Scalable service containers. In *CloudCom*, 2011.