HAL
open science

# Modeling, evaluation and provisioning of elastic service-based business processes in the cloud

Mourad Amziani

## To cite this version:

Mourad Amziani. Modeling, evaluation and provisioning of elastic service-based business processes in the cloud. Other [cs.OH]. Institut National des Télécommunications, 2015. English. NNT : 2015TELE0016 . tel-01217186

**HAL Id: tel-01217186**

**https://theses.hal.science/tel-01217186**

Submitted on 19 Oct 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DOCTORAT EN CO-ACCREDITATION
TELECOM SUDPARIS ET L'UNIVERSITE EVRY VAL D'ESSONNE

**Spécialité: Informatique**

**Ecole doctorale:** Sciences et Ingénierie

**Présentée par**

**Mourad Amziani**

**Pour obtenir le grade de**
**DOCTEUR DE TELECOM SUDPARIS**

# Modeling, Evaluation and Provisioning of Elastic Service-based Business Processes in the Cloud

**Soutenue le 12/06/2015**

**devant le jury composé de** :

*Directeurs de thèse :*

| | |
|---|---|
| Pr. Samir Tata | Télécom SudParis, Institut Mines-Télécom, France |
| Dr. Tarek Melliti | Université d'Evry Val Essonne, France (Encadrant) |

*Rapporteurs :*

| | |
|---|---|
| Pr. Pascal Poizat | Université Paris Ouest Nanterre la Defense, France |
| Pr. Nejib Ben Hadj-Alouane | Ecole Nationale d'Ingénieurs de Tunis, Tunisie |

*Examinateurs :*

| | |
|---|---|
| Pr. Serge Haddad | ENS de Cachan, France (Président du jury) |
| Dr. Kaïs Klai | Université Paris 13, France |
| Dr. Nikolaos Georgantas | INRIA Paris - Rocquencourt, France |

Thèse numéro : 2015TELE0016

# Remerciements

Tout d'abord, je tiens à exprimer toute ma reconnaissance et ma gratitude à mon Directeur de thèse Samir Tata ainsi qu'à mon encadrant Tarek Melliti. Je les remercie de m'avoir encadré, encouragé, aidé et conseillé tout au long de ces quatre dernières années.

Je tiens aussi à remercier les membres du Jury d'avoir accepté de juger mon travail de thèse. Un grand merci à Serge Haddad de m'avoir accordé l'honneur d'être le président de mon jury. Je remercie également les rapporteurs Pascal Poizat et Nejib Ben Hadj-Alouane pour leur lecture attentive de mon manuscrit et leurs nombreux commentaires. Je souhaite aussi remercier Nikolaos Georgantas et Kaïs Klai d'avoir accepté d'examiner mon travail.

Un grand merci aussi à l'ensemble des membres des équipes ACMES (laboratoire Samovar - Télécom SudParis) et COSMO (laboratoire IBISC - Université d'Evry Val d'Essonne).

Pour finir, tous mes remerciements à ma famille pour leur soutien indéfectible et leurs encouragements durant mes longues années d'études. Toute ma reconnaissance à mes très chers parents, Asmane et Ferroudja, qui ont toujours été là pour moi. Je ne pourrais pas finir sans remercier celle qui a eu la patience de me supporter et de m'accompagner durant toutes ces années de thèse, ma chère et tendre Amel.

# Résumé

Le Cloud Computing est de plus en plus utilisé pour le déploiement et l'exécution des applications métiers et plus particulièrement des applications à base de services (AbSs). L'élasticité à différents niveaux est l'une des propriétés fournies par le Cloud. Son principe est de garantir la fourniture des ressources nécessaires et suffisantes pour la continuité de l'exécution optimale des services Cloud. La fourniture des ressources doit considérer la variation de la demande pour éviter la sous-utilisation et la sur-utilisation de ces dernières. Il est évident que la fourniture d'infrastructures et/ou de plateformes élastiques n'est pas suffisante pour assurer l'élasticité des applications métiers déployées. En effet, il est aussi nécessaire de considérer l'élasticité au niveau des applications. Ceci permet l'adaptation dynamique des applications déployées selon la variation des demandes. Par conséquent, les applications métiers doivent être fournies avec des mécanismes d'élasticité permettant leur adaptation tout en assurant les propriétés fonctionnelles et non-fonctionnelles désirées.

Dans nos travaux, nous nous sommes intéressés à la fourniture d'une approche holistique pour la modélisation, l'évaluation et la mise en oeuvre des mécanismes d'élasticité des AbSs dans le Cloud. En premier lieu, nous avons proposé un modèle formel pour l'élasticité des AbSs. Pour cela, nous avons modélisé les AbSs en utilisant les réseaux de Petri et défini deux opérations d'élasticité (la duplication et la consolidation). En outre, nous avons proposé de coupler ces deux opérations avec un contrôleur d'élasticité. Pour assurer l'élasticité des AbSs, le contrôleur analyse l'exécution des AbSs et prend des décisions sur les opérations d'élasticité (duplication/consolidation). Après la définition de notre modèle pour l'élasticité des AbSs, nous nous sommes intéressés à l'évaluation de l'élasticité avant de l'implémenter dans des environnements Cloud réels. Pour cela, nous avons proposé d'utiliser notre contrôleur d'élasticité comme un Framework pour la validation et l'évaluation de l'élasticité en utilisant des techniques de vérification et de simulation. Enfin, nous avons mis en oeuvre l'élasticité des AbSs dans des environnements Cloud réels. Pour cela, nous avons proposé deux approches. La première approche encapsule les AbSs non-élastiques dans des micro-conteneurs, étendus avec nos mécanismes d'élasticité, avant de les déployer sur des infrastructures Cloud. La seconde approche intègre notre contrôleur d'élasticité dans une infrastructure autonomique afin de permettre l'ajout dynamique des fonctionnalités d'élasticité aux AbSs déployées sur des plateformes Cloud.

**Mots-clés**: Cloud Computing, Elasticité, Application à base de services, méthodes formels.

# Abstract

Cloud computing is being increasingly used for deploying and executing business processes and particularly Service-based Business Processes (SBPs). Among other properties, Cloud environments provide elasticity at different scopes. The principle of elasticity is to ensure the provisioning of necessary and sufficient resources such that a Cloud service continues running smoothly even when the number or quantity of its utilization scales up or down, thereby avoiding under-utilization and over-utilization of resources. It is obvious that provisioning of elastic infrastructures and/or platforms is not sufficient to provide elasticity of deployed business processes. In fact, it is also necessary to consider the elasticity at the application scope. This allows the adaptation of deployed applications during their execution according to demands variation. Therefore, business processes should be provided with elasticity mechanisms allowing their adaptation to the workload changes while ensuring the desired functional and non-functional properties.

In our work, we were interested in providing a holistic approach for modeling, evaluating and provisioning of elastic SBPs in the Cloud. We started by proposing a formal model for SBPs elasticity. To do this, we modeled SBPs using Petri nets and defined two elasticity operations (duplication/consolidation). In addition, we proposed to intertwine these elasticity operations with an elasticity controller that monitors SBPs execution, analyzes monitoring information and executes the appropriate elasticity operation (duplication/consolidation) in order to enforce the elasticity of SBPs. After facing the challenge of defining a model and mechanisms for SBPs elasticity, we were interested in the evaluation of elasticity before implementing it in real environments. To this end, we proposed to use our elasticity controller as a framework for the validation and evaluation of elasticity using verification and simulation techniques. Finally, we were interested in the provisioning of elasticity mechanisms for SBPs in real Cloud environments. For this aim, we proposed two approaches. The first approach packages non-elastic SBPs in micro-containers, extended with our elasticity mechanisms, before deploying them in Cloud infrastructures. The second approach integrates our elasticity controller in an autonomic infrastructure to dynamically add elasticity facilities to SBPs deployed on Cloud platforms.



**Keywords**: Cloud Computing, Elasticity, Service-based Business Processes, Formal methods.

# Table of contents


**1 Introduction** **13**
1.1 Context . . . 13
1.2 Motivation And Problem Statement . . . 14
1.3 Objectives and Approach . . . 16
1.3.1 Formal Model for SBPs Elasticity . . . 16
1.3.2 Evaluation of Elasticity Strategies . . . 17
1.3.3 Provisioning of Elasticity in the Cloud . . . 17
1.4 Publications . . . 18
1.5 Thesis Structure . . . 19

**2 Background & State of the Art** **21**
2.1 Introduction . . . 21
2.2 Background . . . 21
2.2.1 Cloud Computing . . . 22
2.2.2 Service-Oriented Architecture . . . 23
2.2.3 Service-based Business Processes . . . 24
2.2.4 Elasticity . . . 25
2.3 State of the Art on Elasticity . . . 27
2.3.1 Analysis Criteria . . . 27
2.3.2 Models and Mechanisms of Elasticity . . . 28
2.3.3 Evaluation of Elasticity Strategies . . . 34
2.3.4 Provisioning of Elasticity Mechanisms in the Cloud . . . 38
2.3.5 Synthesis of Related Work . . . 42
2.4 Conclusion . . . 44

**3 Formal Model for SBPs Elasticity** **45**
3.1 Introduction . . . 45
3.2 Modeling Elasticity of SBPs . . . 46
3.2.1 Illustrating Example . . . 46
3.2.2 Elasticity Mechanisms . . . 47
3.2.3 Modeling Requirements . . . 48
3.3 Modeling of Stateless SBPs Elasticity . . . 50
3.3.1 Stateless SBP Modeling . . . 50
3.3.2 Elasticity Operations . . . 52
3.3.3 Correctness of Elasticity Operations . . . 55
3.4 Modeling of Stateful SBPs Elasticity . . . 58
3.4.1 Stateful SBP Modeling . . . 58
3.4.2 Elasticity Operations . . . 61
3.5 Modeling of Timed SBPs Elasticity . . . 63

3.5.1 Timed SBP Modeling . . . 63
3.5.2 Elasticity Operations . . . 65
3.6 Generic Controller for SBPs Elasticity . . . 68
3.6.1 Elasticity Controller . . . 68
3.6.2 Formal Description of the Generic Controller . . . 70
3.6.3 Illustrating Example of SBPs elasticity . . . 72
3.7 Conclusion . . . 75

**4 Evaluation of Elasticity Strategies 77**
4.1 Introduction . . . 77
4.2 Generic Controller for the Evaluation of Elasticity Strategies . . . 78
4.3 Validating Elasticity Strategies . . . 80
4.3.1 Verification-based Evaluation . . . 80
4.3.2 Proof of Concept: Validating Elasticity Strategies . . . 81
4.4 Comparing Elasticity Strategies . . . 87
4.4.1 Performance-based Evaluation . . . 88
4.4.2 Reliability-based Evaluation . . . 88
4.4.3 Proof of Concept: Comparing Elasticity Strategies . . . 89
4.5 Conclusion . . . 98

**5 Provisioning of Elasticity Mechanisms in the Cloud 99**
5.1 Introduction . . . 99
5.2 Provisioning Requirements . . . 100
5.3 Provisioning of Elastic SBPs on Cloud infrastructures . . . 102
5.3.1 Service micro-containers . . . 102
5.3.2 Elastic Service micro-containers . . . 104
5.3.3 Experiment . . . 106
5.4 Provisioning of Elastic SBPs on Cloud platforms . . . 111
5.4.1 Autonomic Management for OCCI Resources . . . 112
5.4.2 Autonomic Approach for SBPs elasticity . . . 115
5.4.3 Experiment . . . 119
5.5 Conclusion . . . 123

**6 Conclusion 125**
6.1 Contributions . . . 125
6.2 Perspectives . . . 126
6.2.1 Short-term perspectives . . . 126
6.2.2 Long-term perspectives: Mixing Multi-tenancy and Elasticity of Business Processes . . . 127

# List of Figures


2.1 Cloud Computing Conceptual Reference Model defined by NIST [1]. . 22
2.2 SOA architecture . . . . . 24
2.3 The principle of elasticity . . . . . 25
2.4 Two-Tier SaaS Scaling and Schedule Architecture [2]. . . . . 29
2.5 Architecture for elastic resource provisioning of BPEL workflows [3]. . 31
2.6 Vadara's Architecture for Cloud applications elasticity [4]. . . . . 32
2.7 Service provider architecture for elastic resource provisioning [5]. . . . 33
2.8 The analytical model for elasticity evaluation [6]. . . . . 36
2.9 Service provider architecture for elastic resource provisioning [7]. . . . 37
2.10 Modeling Cloud service behavior process [8]. . . . . 37
2.11 ViePEP Architecture [9] . . . . . 39
2.12 Extending the OpenNebula architecture with a Forecasting engine [10]. 40
2.13 ElaaS Components [11] . . . . . 40
2.14 The QoS-Aware Resource Elasticity (QRE) framework [12] . . . . . 41
2.15 Planification component for the elasticity of Cloud applications [13] . . 43

3.1 BPMN model of the SBP example of the online computer shopping service. . . . . 47
3.2 SBP execution model of the online shopping service . . . . . 48
3.3 Petri net of the SBP of the online computer shopping service. . . . . 49
3.4 Example of the elasticity of a SBP . . . . . 54
3.5 An example of the elasticity of the stateful SBP of the online computer shopping service. . . . . 62
3.6 An example of the elasticity of the timed SBP of the online computer shopping service. . . . . 67
3.7 An overview of the Control Loop for SBPs elasticity. . . . . 69
3.8 HLPN model of the generic controller . . . . . 71
3.9 The initial SBP managed by the elasticity Controller . . . . . 73
3.10 The SBP at time $t1$. . . . . 73
3.11 Duplication of the service S2 at time $t2$. . . . . 74
3.12 The SBP at time $t3$. . . . . 74
3.13 Consolidation of the service S2 at time $t4$. . . . . 75

4.1 Generic Controller for the Evaluation of Elasticity Strategies . . . . . 79
4.2 An example of the SNAKES reachability graph of an elasticity strategy 82
4.3 Petri net of the SBP of the online computer shopping service. . . . . 84
4.4 The average evolution of resources consumption with strategy 1 . . . . 90
4.5 The average evolution of resources consumption with strategy 2 . . . . 90
4.6 The evolution of average response time according to the average of demands . . . . . 91

4.7 The evolution of resources consumption according to the demand of resources . . . 91
4.8 Petri net of the Timed SBP of the online computer shopping service. . 92
4.9 the evolution of the minimal calls processing time according to load variation . . . 93
4.10 the evolution of the average calls processing time according to load variation . . . 94
4.11 the evolution of the proportion of lost calls according to load variation 94

5.1 Elasticity approach for SBPs elasticity. . . . 101
5.2 Service micro-container packaging framework. . . . 103
5.3 Extended service micro-container packaging framework. . . . 105
5.4 Service micro-containers (a) duplication and consolidation (b) steps. . 106
5.5 An example of the provisioning of an elastic SBP . . . 107
5.6 Calls arrival scenarios for invoking the SBP . . . 108
5.7 Response time of the SBP before and after adding elasticity with the scenario 1. . . . 109
5.8 Memory consumption of the SBP before and after adding elasticity with the scenario 1. . . . 110
5.9 Response time of the SBP before and after adding elasticity with the scenario 2. . . . 110
5.10 Memory consumption of the SBP before and after adding elasticity with the scenario 2. . . . 111
5.11 Autonomic control loop for a Cloud resource . . . 112
5.12 OCCI extension for Autonomic Computing. . . . 114
5.13 Autonomic Computing Infrastructure establishment for SBP elasticity. 116
5.14 Response time of the SBP before and after adding our Autonomic Infrastructure. . . . 122
5.15 Memory consumption of the SBP before and after adding our Autonomic Infrastructure. . . . 122

# List of Tables

4.1 Verification-based evaluation of elasticity strategies. . . . . 87

Chapter 1

# Introduction

Contents


**1.1 Context** . . . . . **13**
**1.2 Motivation And Problem Statement** . . . . . **14**
**1.3 Objectives and Approach** . . . . . **16**
1.3.1 Formal Model for SBPs Elasticity . . . . . 16
1.3.2 Evaluation of Elasticity Strategies . . . . . 17
1.3.3 Provisioning of Elasticity in the Cloud . . . . . 17
**1.4 Publications** . . . . . **18**
**1.5 Thesis Structure** . . . . . **19**


## 1.1 Context

Cloud computing is a new delivery model, based on the pay-as-you-go business principle, for IT services [14]. Cloud computing is now recognized as an effective paradigm for developing and delivering services over the Internet. It typically involves provisioning dynamically scalable and often virtualized resources. It delivers services at the layers of infrastructure, platform and software. Cloud services use Cloud components (such as databases, containers, VMs etc.) which themselves use Cloud resources (such as CPU, memory, network). Adoption of Cloud computing reduces the maintenance complexity and cost for enterprise customers, and provides on-going revenue for providers [15]. More and more companies are adopting the new economic model offered by Cloud computing. For instance, a survey conducted by the Cloud Industry Forum [16] in 2012, involving 300 companies, shows that 53% of the companies are currently adopting the Cloud. The same survey showed that 73% of them are planning to increase their adoption of Cloud services in the next 12 months.

Cloud environments are being increasingly used for deploying and executing business processes and particularly service-based business processes [17]. A Service-based Business Process (SBP) is a business process that consists in assembling a set of elementary IT-enabled services which are related in terms of their contribution to the

overall realization of the business process. As it has been the case with other technologies, the availability of Business Process Management (BPM) in the Cloud allows imagining new usage scenarios. Typically, these scenarios include the execution of thousands of processes (set of coordinated activities) during a very short period of time requiring temporarily a very important amount of resources. Other than getting new usage scenarios, Cloud users can benefit from Cloud infrastructures to execute innovative solution in scenarios like political campaign management or crisis management where the need for quick adaptation is at the essence but at the cost of very sophisticated development. Novel and innovative approaches for modeling, deploying and enactment of business processes should be developed to allow supporting the scenario cited above and others in a safer and cost-effective way.

Among other properties, Cloud environments provide elasticity at different scopes. According to Dustdar et al. [18], one of the key features driving the popularity of Cloud computing is elasticity: resources must be managed by scaling up and down as needed so that limited resources can be offered for potentially unlimited utilization. The principle of elasticity is to ensure the provisioning of necessary and sufficient resources such that a Cloud service continues running smoothly even when the number or quantity of its utilization scales up or down, thereby avoiding under-utilization and over-utilization of resources [19]. Provisioning of resources can be made using vertical or horizontal elasticity [20]. Vertical elasticity increases or decreases resources consumed by Cloud services while the horizontal elasticity replicates or removes instances of Cloud services [11]. Elasticity can play an important role to respect the agreements between the service's user and its provider. From the user's perspective, the elasticity ensures an efficient provisioning of resources which maintains the QoS (e.g., minimizing task execution time), without exceeding a given budget. From the provider's perspective, the elasticity maximizes the financial gain by ensuring better use of computing resources and allowing multiple customers to be served simultaneously while keeping these customers satisfied [8, 21].

## 1.2 Motivation And Problem Statement

The adoption of Cloud computing could increase rapidly if providers prove their ability to continuously ensure the required QoS of deployed business processes [22]. Indeed, this is a challenging task because these latter are exposed to dynamic evolution and workload fluctuation during their life-cycle. In order to preserve the QoS of deployed business processes in such an environment, elasticity mechanisms should be offered by Cloud providers in order to enable dynamic adaptation of resources consumed with minimal cost and performance degradation.

It is obvious that provisioning of elastic infrastructures (e.g., based on elasticity of virtual machines) and/or platforms (e.g. based on elasticity of process engines or service containers), is not sufficient to provide elasticity of deployed business processes. In fact, it is also necessary to consider the elasticity at the application scope. This

allows the adaptation of deployed applications during their execution according to demands variation. Therefore, business processes should be provided with elasticity mechanisms allowing their adaptation to the workload changes while ensuring the desired functional and non-functional properties.

When applied to SBPs, elasticity becomes a complicated problem with many issues to be addressed. In this thesis we address elasticity of SBPs in the Cloud that mainly raises the following questions:

- What model and mechanisms should be defined and developed to describe and ensure SBPs elasticity?
- How to evaluate SBPs elasticity before implementing it in real Cloud environments?
- How to provision elasticity mechanisms for SBPs in real Cloud environments?

To describe SBPs elasticity, we need to have a model that allows the representation of SBPs as well as the different reconfigurations caused by the execution of elasticity actions. We argue that it is beneficial to adopt formal models to describe SBPs elasticity which provides rigorous description and allows verification of properties. Ensuring SBPs elasticity can be done using vertical or horizontal elasticity mechanisms. Vertical mechanisms have to add/remove as resources as necessary to a SBP by re-engineering it and/or its container. Horizontal mechanisms have to create as copy as necessary of a SBP by duplicating the process or a part of it.

Evaluating SBPs elasticity consists in evaluating the strategies used for ensuring the elasticity. An elasticity strategy is responsible of making decisions on the execution of elasticity mechanisms i.e., deciding when, where and how to use elasticity mechanisms. Considering the abundance of possible strategies, it is necessary to evaluate and validate the correctness and performance of these strategies before using them in real Cloud environments.

Any proposed approach for ensuring SBPs elasticity in the Cloud needs to be validated in realistic environments to ensure its effectiveness in enhancing SBPs behaviors. To this end, it is necessary to provide mechanisms for the provisioning of SBPs elasticity in real Cloud environments while considering both provisioning contexts: Infrastructure and Platform.

Many attempts to provide elasticity of SBPs have been proposed, but as we will explain, almost all of them deal with the elasticity at the infrastructure scope. Furthermore, most of the existing proposals related to elasticity at the application scope are not suitable for SBPs since they do not consider the nature of the SBPs (i.e., elasticity at the scope of services). In addition, existing approaches for elasticity evaluation are limited to quantitative evaluation of strategies and do not allow formal verification of the correctness of the elasticity strategies. Finally, approaches for the provisioning of elasticity mechanisms in the Cloud require changing the nature of the SBPs and/or actual Cloud environments.

## 1.3 Objectives and Approach

Our work is mainly concerned with providing horizontal elasticity for SBPs. This thesis does not discuss all the aspects related to elasticity. For example, we do not deal with ensuring vertical elasticity of SBPs or managing the underlying Cloud infrastructure. While we believe that a lot of issues related to elasticity are important to address, the provisioning of horizontal elasticity of SBPs discussed here is complex enough to deserve a separate treatment.

We aim in this thesis at proposing an approach for ensuring horizontal elasticity of SBPs in the Cloud. To do so, we propose to answer the three questions we raised above by:

- Defining a formal model for describing and ensuring SBPs elasticity.
- Providing a framework for the validation and evaluation of SBPs elasticity.
- Providing mechanisms for the provisioning of elastic SBPs in real Cloud environments.

### 1.3.1 Formal Model for SBPs Elasticity

Performing horizontal elasticity can be ensured by providing Cloud environments with elasticity mechanisms that allow deployed SBPs to scale up and down. In order to manage these elasticity mechanisms, two approaches can be used. 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. However, this approach changes the nature of the deployed SBPs. The second approach that we propose to adopt in our work consists in setting up a controller that continuously monitors SBPs execution, analyzes monitoring information and executes appropriate actions (duplication/consolidation) in order to ensure the QoS of deployed SBPs while avoiding over-provisioning and under-provisioning of resources.

In our work, we propose a formal model for SBPs elasticity that intertwines two elasticity operations (duplication/consolidation) with a generic controller for elasticity. Our goal is to provide a model for describing SBPs elasticity and mechanisms for ensuring elasticity of SBPs at the scope of services.

To this end, we propose a SBP model, based on Petri nets, to describe SBPs while allowing the representation of the notion of basic services. Using this SBP model, we are able to define and formalize elasticity operations (duplication/consolidation) that operate at the scope of services. These operations allow ensuring the SBP elasticity by duplicating only services that are overloaded (bottlenecks). Furthermore, we prove the correctness of these two elasticity operations on preserving the semantics of SBPs. In addition, we show that, using our formal model, the elasticity of stateless, stateful and timed SBPs can be ensured. Moreover, we propose to model the elasticity

controller using high level Petri nets. This controller is generic in order to allow the implementation and execution of different elasticity strategies.

### 1.3.2 Evaluation of Elasticity Strategies

Elasticity strategies are responsible of making decisions on the execution of elasticity mechanisms i.e., deciding when, where and how to use duplication/consolidation operations. These strategies are responsible of guaranteeing provisioning of necessary and sufficient resources that ensure a smooth functioning of deployed SBPs despite of changes in SBPs solicitations. Many strategies can be proposed to ensure SBPs elasticity. The abundance of possible strategies requires their evaluation and validation in order to guarantee their effectiveness before using them in real Cloud environments.

In our work, we propose an evaluation approach that uses the previously defined elasticity controller as a framework for the evaluation of different elasticity strategies. Our goal is to show how we can perform, using our generic controller, validation and comparison of elasticity strategies.

To evaluate elasticity strategies, we propose two evaluation approaches: The first approach allows the validation of elasticity strategies using a verification technique. The second approach allows the comparison between elasticity strategies using a simulation technique. On one hand, the verification-based approach allows performing qualitative evaluations of elasticity strategies by using formal methods (e.g., model-checking techniques). On the other hand, the simulation allows performing quantitative evaluations of elasticity strategies by calculating performance indicators (e.g., resource consumption, time response) related to SBPs elasticity.

### 1.3.3 Provisioning of Elasticity in the Cloud

In order to provision elastic SBPs in the Cloud, we propose to go further in our work by using our elasticity mechanisms in real Cloud environments. Our goal is to show how we can use the elasticity mechanisms we have previously developed in order to provision SBPs elasticity in real Cloud environments without changing the nature of the SBPs or existing Cloud environments.

To this end, we propose two approaches for the provisioning of elastic SBPs while considering two provisioning contexts: Cloud infrastructures and Cloud platforms. The first approach consists in an end-to-end approach for the provisioning of elastic SBPs on Cloud infrastructures. This approach packages non-elastic SBPs in micro-containers, extended with our elasticity mechanisms, before deploying them in real Cloud infrastructures. The second approach that we propose takes advantage of existing PaaS mechanisms in order to ensure SBPs elasticity on Cloud platforms. In this approach, we propose to integrate our elasticity controller in an autonomic loop for Cloud resources in order to dynamically add elasticity facilities to SBPs in real Cloud platforms.

Any proposed solution for ensuring SBPs elasticity in the Cloud needs to be experimented and validated to ensure its effectiveness in enhancing SBPs behaviors. In our work we experiment and validate our approaches in a realistic Cloud environment. The validation includes the implementation aspects as well as the experiments and their results.

## 1.4 Publications

This research led to the following publications:

**Modeling SBPs Elasticity**

- Mourad Amziani, Tarek Melliti and Samir Tata. A Generic Framework for Service-based Business Process Elasticity in the Cloud (short paper). In the 10th International Conference on Business Process Management, BPM 2012, Estonia, September 2012

- Mourad Amziani, Tarek Melliti and Samir Tata. Formal Modeling and Evaluation of Service-based Business Process Elasticity in the Cloud. In the 22nd International Conference on Collaboration Technologies and Infrastructure, IEEE WETICE 2013, Tunisia, June 2013

**Evaluating SBPs Elasticity**

- Mourad Amziani, Tarek Melliti and Samir Tata. Formal Modeling and Evaluation of Stateful Service-based Business Process Elasticity in the Cloud. In the 21st International Conference on Cooperative Information Systems, CoopIS 2013, Austria, September 2013

- Mourad Amziani, Kaïs Klai, Tarek Melliti and Samir Tata. Time-based Evaluation of Service-based Business Process Elasticity in the Cloud. In the 5th IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2013, United Kingdom, December 2013

**Provisioning of Elastic SBPs in the Cloud**

- Mohamed Mohamed, Mourad Amziani, Djamel Belaïd, Samir Tata and Tarek Melliti. An Autonomic Approach to Manage Elasticity of Business Processes in the Cloud. In Future Generation Computer Systems (Elsevier), October 2014, ISSN 0167-739X.

## 1.5 Thesis Structure

The remainder of this manuscript is organized as follows. Chapter 2 contains a description of the different concepts needed for understanding the rest of the work as well as the state of the art on elasticity. In this chapter, we give the definitions of Cloud Computing, SOA and Service-based Business Processes as well as Elasticity. Furthermore, we give an overview of the state of the art related to elasticity. Therein, we will cite the different works dealing with elasticity with highlights on the advantages that we would like to have in our solution and the drawbacks that we would like to avoid.

In chapter 3, which is dedicated to the first question we raised in the problem statement, we present our proposals to model SBPs elasticity. The proposal entails our elasticity approach to ensure elasticity at the scope of services and a formal model that intertwines two elasticity operations (duplication/consolidation) with a generic controller to ensure SBPs elasticity.

In chapter 4, which is dedicated to the second question we raised in the problem statement, we push further our work by proposing to use the elasticity controller as a framework for the evaluation of different elasticity strategies. In our approach we propose to use two techniques for the evaluation of elasticity strategies: verification and simulation.

In chapter 5, which is dedicated to the third question we raised in the problem statement, we propose two approaches for the provisioning of elastic SBPs in real Cloud environments. The first approach consists in an end-to-end approach for the provisioning of elastic SBPs on Cloud infrastructures. The second approach consists in an autonomic infrastructure that dynamically adds elasticity facilities to SBPs on Cloud platforms.

Finally, we sum up our contributions in chapter 6. We conclude this last chapter with some perspectives that we aim to tackle in future endeavours.

Chapter 2

# Background & State of the Art

**Contents**


**2.1 Introduction** . . . . . **21**
**2.2 Background** . . . . . **21**
2.2.1 Cloud Computing . . . . . 22
2.2.2 Service-Oriented Architecture . . . . . 23
2.2.3 Service-based Business Processes . . . . . 24
2.2.4 Elasticity . . . . . 25
**2.3 State of the Art on Elasticity** . . . . . **27**
2.3.1 Analysis Criteria . . . . . 27
2.3.2 Models and Mechanisms of Elasticity . . . . . 28
2.3.3 Evaluation of Elasticity Strategies . . . . . 34
2.3.4 Provisioning of Elasticity Mechanisms in the Cloud . . . . . 38
2.3.5 Synthesis of Related Work . . . . . 42
**2.4 Conclusion** . . . . . **44**


## 2.1 Introduction

In this thesis we aim to provide elasticity mechanisms for Service-based Business Processes (SBPs) in Cloud environments. In order to understand the remainder of this manuscript, one should have a basic knowledge on different paradigms and concepts related to our work. We dedicate the first section of this chapter to briefly introduce these basics to the reader. In the second section, we discuss a selection of works dealing with elasticity in Cloud environments.

## 2.2 Background

In this section, we introduce definitions and basics that are related to our work. To do so, we start by presenting the context of our work which is Cloud Computing. Afterwards, we introduce the Service-Oriented Architecture and Service-based Business Processes that represent the specific kind of applications that we basically target. Finally, we define elasticity aspects we aim to deal with in our work.

### 2.2.1 Cloud Computing

Cloud computing is an emerging paradigm in information technology. It is defined by the National Institute of Standards and Technology (NIST) [1] as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Figure 2.1 shows an overview of Cloud computing reference architecture viewed by NIST. It identifies the major actors, their activities and functions in the Cloud. This Figure aims to facilitate the understanding of the requirements, uses, characteristics and standards of Cloud computing.

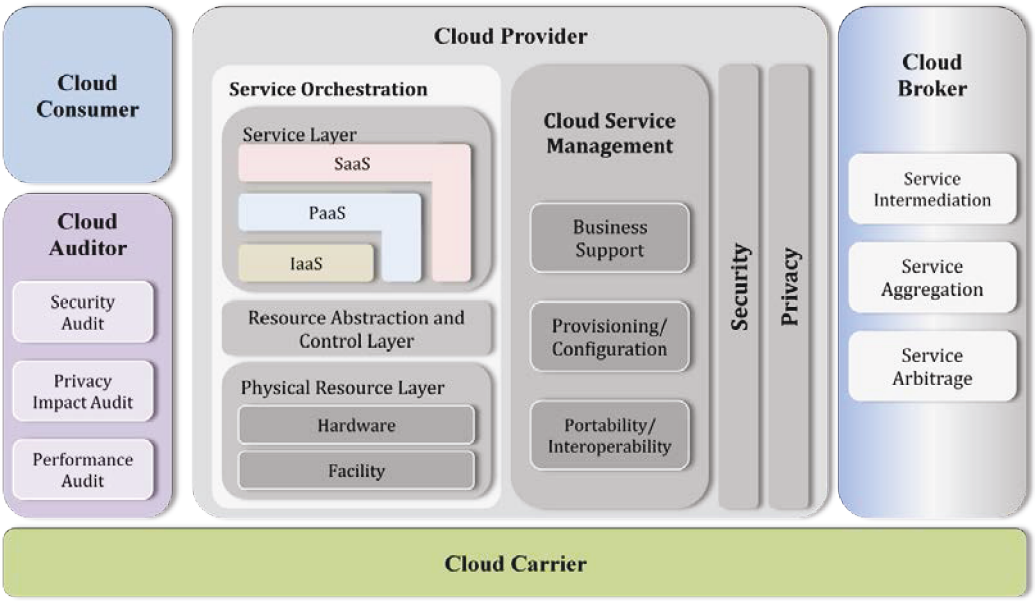


Figure 2.1: Cloud Computing Conceptual Reference Model defined by NIST [1].

Cloud computing is characterized by its economic model based on "pay-as-you-go" model. This model allows a user to consume computing resources as needed. Moreover, resources in the Cloud are accessible over the network through standard mechanisms that promote use by different platforms. The resources are offered to consumers using a multi-tenant model with heterogeneous resources assigned to consumer on demand. These resources are provisioned in an elastic manner that allows scaling up or down rapidly commensurate with demand. Furthermore, Cloud resources usage can be monitored and controlled in order to respect the "pay-as-you-go" model.

Services in the Cloud are basically delivered under three well discussed layers namely the Infrastructure as a Service (IaaS), the Platform as a Service (PaaS) and the Software as a Service (SaaS). Nowadays, more services appeared called generally as XaaS that target a specific area. For example there is the DaaS for Desktop as a

Service, NaaS for Network as a Service, etc.

- **IaaS:** Consumers are able to use infrastructure resources. These resources can be virtual machines, storage resources and networks. The provider takes the responsibility of installing, managing and maintaining these resources transparently.

- **PaaS:** Consumers are able to develop, deploy and manage their applications onto the Cloud using the libraries, editors and services offered by the provider. The provider takes the responsibility to provision, manage and maintain the Infrastructure resources.

- **SaaS:** Consumers are able to use running applications on a IaaS or PaaS through an interface. They are not responsible of managing or maintaining the used Cloud infrastructure and/or platfrom .

Clouds can be provisioned following different models according to the user's needs. If the Cloud is used by a single organization, we talk about Private Cloud. In this case, this organization owns the Cloud and is responsible of its management and maintenance. However, if the Cloud is owned by different organizations, we are talking about community or federation Cloud. Whenever the Cloud is exposed to public use, we are talking about Public Cloud. In this case, an organization owns the Cloud and manages it while it is used by other consumers. Finally, there is another model in which the Cloud is composed of two or more Clouds called Hybrid Clouds. In these Clouds there are Public or Private Clouds glued together.

### 2.2.2 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is the outcome of the Web services developments and standards in support of automated business integration [23] [24]. The purpose of this architecture style is to address the requirements of loosely coupled, standards-based, and protocol-independent distributed computing. As defined by Papazoglou [25], *SOA is a logical way of designing a software system to provide services to either end user applications or other services distributed in a network through published and discoverable interfaces.*

The main building blocks in SOA are services. Services are self-describing components that support rapid, low-cost development and deployment of distributed applications. Thus, using SOA, applications are defined as a composition of re-usable software services, which implement the business logic of the application domain.

The SOA architectural style is structured around the three basic actors depicted in Figure 2.2: Service Provider, Service Client and Service Registry while the interactions between them involve publish, find and bind operations. Service Provider is the role assumed by a software entity offering a service. Service Client is the role of a requestor

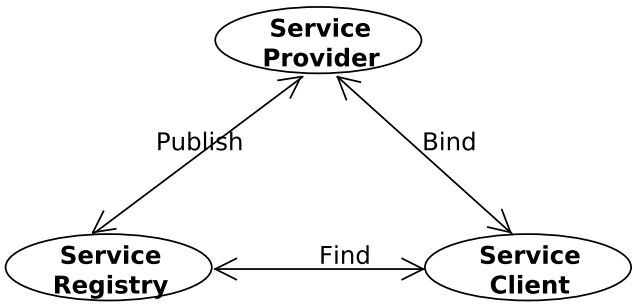


Figure 2.2: SOA architecture

entity seeking to consume a specific service. However, Service Registry is the role of an entity maintaining information on available services and the way to access them.

The benefit of this approach lies in the loose coupling of the services making up an application. Services are provided by platform independent parts, implying that a client using any computational platform, operating system and any programming language can use the service. While different Service Providers and Service Clients may use different technologies for implementing and accessing the business functionality, the representation of the functionalities on a higher level (services) is the same. Therefore, it should be interesting to describe an application as a composition of services in order to be implementation-agnostic. This allows the separation of the business functionality from its implementation. Hence, an application can be executed by composing different services provided by heterogeneous parts with respect to their services descriptions.

### 2.2.3 Service-based Business Processes

A Business Process is a set of one or more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships. A process may be wholly contained within a single organizational unit or may span several different organizations, such as in a customer-supplier relationship [26].

A Service-based Business Process (SBP) is a business process that consists in assembling a set of elementary IT-enabled services which are related in terms of their contribution to the overall realization of the business process. A service is typically the smallest unit of work that represents a module offering computation or data capabilities. 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).

SBPs can be designed using a wide variety of languages (BPMN, Petri nets, etc.).

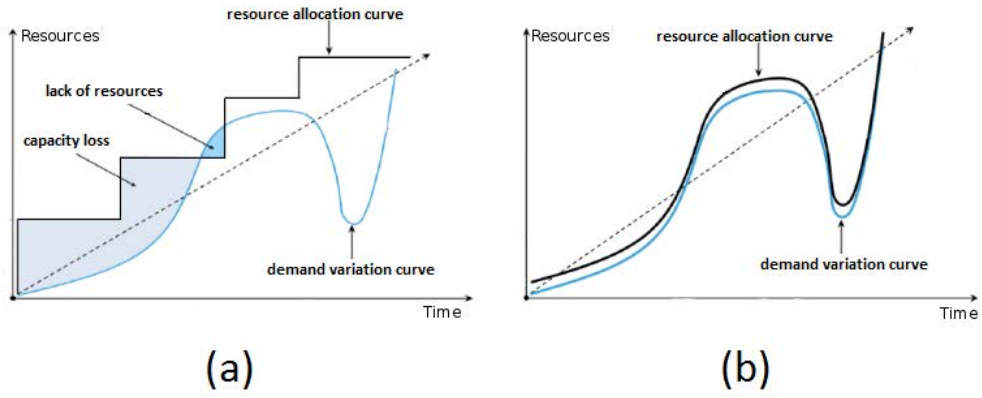


Figure 2.3: The principle of elasticity

Three types of languages can be identified [26]:

- **Formal languages** (Markov chains, Petri nets, etc.): These languages are based on theoretical models. These languages are generally provided with unambiguous semantics and allows analysis techniques (e.g., model checking and simulation) to answer questions related to correctness and performance.
- **Conceptual languages** (BPMN, UML activity diagram, etc.): These languages are typically informal (higher-level languages), i.e., they do not have a well-defined semantics and do not allow analysis.
- **Execution languages** (BPEL, etc.): These languages are "technical" languages that are used for specifying enactment of business processes.

### 2.2.4 Elasticity

In physics, elasticity is a property of an object which returns to its original form after being deformed [27]. In Cloud environments, the principle of elasticity is to ensure the provisioning of necessary and sufficient resources such that a Cloud service continues running smoothly even when the number or quantity of its utilization scales up or down, thereby avoiding under-utilization and over-utilization of resources [19]. According to Dustdar et al. [18], one of the key features driving the popularity of Cloud computing is elasticity: resources must be managed by scaling up and down as needed so that limited resources can be offered for potentially unlimited utilization.

Traditional provisioning of a fixed amount of IT resources is not suitable with high dynamic environments such Cloud environments. Fixing an amount of resource that could handle peak of demands would generate over-provisioning of resource at major period of time which can leads to loss of money. In contrast, fixing an amount of

resources that is not sufficient to handle demands would generate under-provisioning of resources which can leads to loss of QoS. Ensuring elasticity allows having a resource provisioning that automatically adapts to workload change in order to ensure QoS while reducing cost.

If we consider the resources demands shown in Figure 2.3, the resource allocation should allow the provisioning of sufficient resources in order to ensure the satisfaction of demands at each period of time while avoiding the under-utilization of resources. The Figure 2.3-(a) represents an example of a resources allocation approach to manage this kind of demands. As we can see, this resource allocation causes an under-utilization of resources during an important portion of time. In addition, it causes an over-utilization of resources during a certain period of time. The best solution in this case, is to have an allocation of resources that automatically adapts to demands variations while ensuring QoS and reducing cost (see Figure 2.3-(b)). This dynamic adaptation to changes is what we call elasticity.

The elasticity can be defined by a set of characteristics (scope, metric, strategy and method/technique) [21]:

**Scope:** is about the nature of the Cloud service we want to make elastic. It defines where the elasticity actions are executed (Infrastructure, platform, application, data, etc.)

**Metric:** is about the indicators that are used to make decisions about elasticity. These metrics depend on the scope of elasticity we deal with:

- Resource type: storage, CPU, Business service, etc.
- Quantity, Quality: size, frequency, response time, etc.
- Price

**Strategy:** is about the policies that are used to make decisions about elasticity. These strategies can be manual (e.g., ensured by human administrator) or automatic. Manual strategies mean that a human administrator monitors deployed Cloud services and performs elasticity actions when needed. Automatic strategies mean that an elasticity component (e.g., controller) monitors the deployed Cloud service, collects information about its execution, and performs elasticity actions according to user-defined rules (e.g., SLA). These automatic strategies can be of the following types:

- Reactive strategies: are based on rules (Event-Condition-Action mechanisms). When an event occurs and a condition is satisfied, an elasticity action is triggered. The used conditions are generally threshold-based conditions (i.e., once a given threshold is violated, an elasticity action is performed).
- Programmed strategies: are based on scheduled execution of an elasticity action at a certain period of time (e.g., increase an online sales site' resources during the sales period).

- Predictive strategies: are based on predictive-performance models and load forecasts. These strategies aim to anticipate future violations and perform elasticity actions to avoid these violations.
- Hybrid strategies: a combination of a set of reactive, programmed or predictive strategies.

**Method/Technique:** is about reconfiguration actions that are used to ensure the elasticity. These actions can be:

- Duplication/Consolidation: consists of adding or removing instances (copies) of a Cloud service.
- Resizing: consists in adding or removing resources (e.g., CPU, memory or storage) to a Cloud service.
- The execution of these actions can produce consequences like bursting, migration, etc.

Provisioning of resources can be made using vertical or horizontal elasticity [20]. Vertical elasticity increases or decreases resources consumed by Cloud services (e.g., changing the CPU, adapting the memory size or bandwidth of VMs), while the horizontal elasticity replicates or removes instances of Cloud services [11]. On one hand, vertical elasticity can be performed by re-engineering of services/applications (at Software layer) and/or application containers (at Platform layer). On the other hand, horizontal elasticity can be perfomed by providing elasticity mechanisms (duplication/consolidation) that enforce elasticity of deployed SBPs.

## 2.3 State of the Art on Elasticity

In this section, we present a selection of works dealing with elasticity in Cloud and Virtualized environments. In fact, a plethora of works exist in the literature aiming at providing elasticity solutions. However, in this manuscript we refrain from citing all these works to highlight a selection of them that we believe representative.

### 2.3.1 Analysis Criteria

Before presenting these approaches, we present in the following criteria we believe important to consider when analysis the state of the art.

This manuscript is mainly concerned with providing elasticity of Service-based Business Processes in the Cloud. To do this, elasticity should be provided at the SaaS layer. Considering the nature of SBPs, we argue that achieving this goal has to go through providing mechanisms for not only ensuring elasticity at the process scope but also at the service scope. In fact, we think that it is not necessary to operate at

the process scope while the bottlenecks can came from some services that compose the process. In order to provide elasticity at the scope of services, we have to use metrics that are related to services execution to take decision about SBPs elasticity. For instance, we can consider performance indicators such as services response time, services workload, etc.

We argue that it is beneficial to adopt formal models to describe SBPs elasticity. These latter provide rigorous description of SBPs elasticity and allow verification of properties related to elasticity and/or SBPs execution. In fact, since several strategies can be used to make decisions about elasticity, it is necessary to be able to not only compare strategies' performance but also to formally verify the correctness of these strategies before using them in real Cloud environments. In this context, it would be interesting to be able to evaluate and use different strategies. To do this, elasticity mechanisms should be generic in order to allow using different strategies.

In our work, we are interested in providing horizontal elasticity. We do not discuss all the aspects that are relevant to elasticity. For example, we do not deal with ensuring vertical elasticity of SBPs. While we believe that these issues are important, the provisioning of horizontal elasticity of SBPs discussed here is complex enough to deserve separate treatment. To manage SBPs elasticity, there are basically two approaches. 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 elasticity mechanisms [28]. This approach dedicates a controller for each service of the SBP but changes the nature of the considered SBP. The second approach consists in setting up a controller that enforces elasticity of deployed SBPs without changing the nature of these latter. However, it is also necessary to take into account the controller scalability in order to avoid having a single failure point.

In order to provision mechanisms for SBPs elasticity in real Cloud environments, we need to keep in mind that any proposed solution should not change the nature of existing Cloud environments and/or deployed applications. In addition, it should not require any effort from the application developer/administrator in order to facilitate its adoption.

In the following we discuss a selection of works dealing with elasticity in Cloud environments. We start by presenting the works proposing models and mechanisms for ensuring elasticity. Afterwards, we present the state of the art on elasticity evaluation. Then, we present works dealing with provisioning of elasticity mechanisms in Cloud environments. Note here that there are works that could be presented for one or more areas. We conclude this section by a synthesis of the presented works comparing them on the basis of the criteria that we dressed in our research objectives.

### 2.3.2 Models and Mechanisms of Elasticity

As stated before, elasticity can be ensured using mechanisms of duplication/consolidation (i.e., horizontal elasticity). In our work, we are interested in providing elasticity

of Service-based Business Processes (SBPs). To do this, elasticity mechanisms have to be provided at the software (SaaS) layer. Duplication/consolidation mechanisms have been widely considered in the areas of service adaptation (e.g., in order to meet QoS constraint) [29], service availability [30] and fault tolerance [31]. These approaches allow SBPs reconfiguration to respond to the change of workload. Nevertheless, the proposed mechanisms duplicate the entire SBP and so, of all its services (elasticity at the process scope) while the bottleneck can came from a single service or a certain number of services of the SBP. Indeed, the disadvantage of these approaches is the unnecessary resources consumption caused by duplication of all services of the considered SBP. We advocate in our work that it is not necessary to duplicate all the services while the bottleneck can come from one or some services of the SBP. We think that the duplication of only overloaded services can solve the elasticity problem while avoiding unnecessary resources consumption.

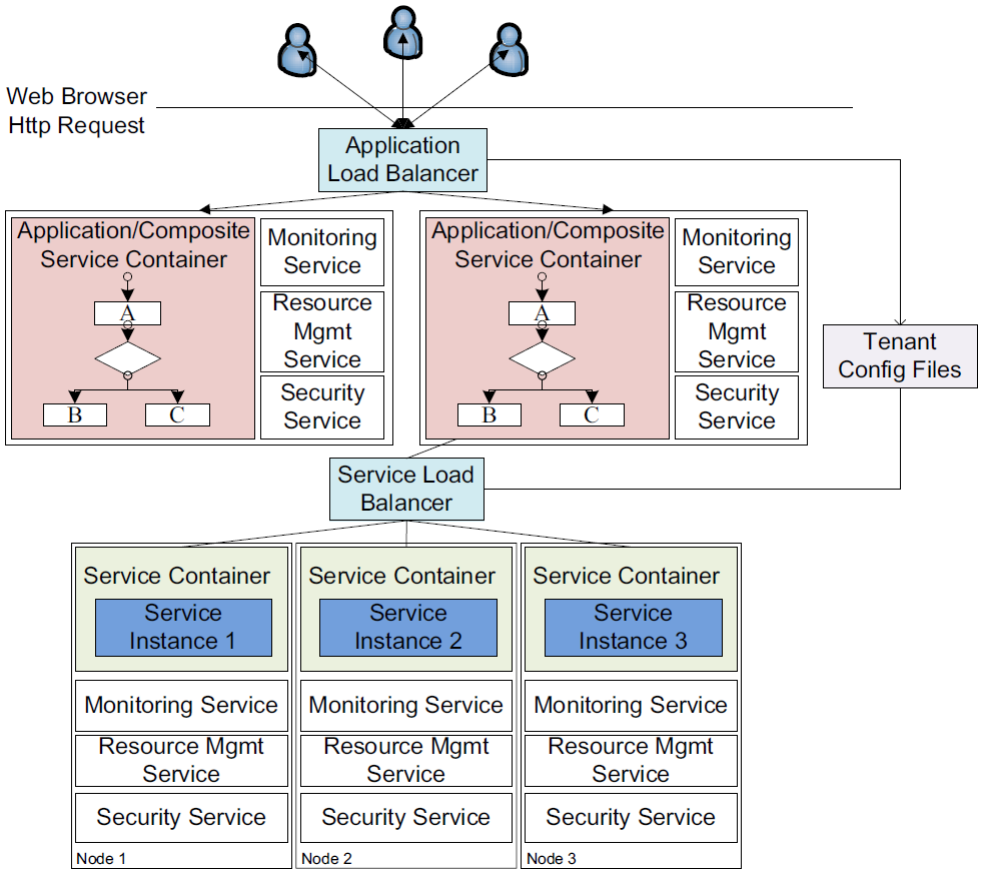


Figure 2.4: Two-Tier SaaS Scaling and Schedule Architecture [2].

In the literature, few works discussed the elasticity at the service scope. In [2],

authors considered scaling at both service and application scopes in order to ensure elasticity. To this end, they proposed a two-tier SaaS scaling and scheduling architecture that allows operating duplication actions at both application and service scopes. In addition, they proposed a cluster-based resource allocation algorithm that selects suitable servers' nodes to run newly duplicated application/service instances. An overview of the proposed architecture is given in Figure 2.4. As shown in this latter, the Application/Service Load Balancer routes application (or service) calls to the different instances. When all the application (or service) instances are overloaded, it duplicates this instances by deploying a new application (or service) instance on a new server node. The proposed architecture is composed of a set of components: The Service Container component is the runtime environment for an application/service. This component includes monitoring service, resource management service and security service. The re-deployable Service Package component is a package that contains the source and compiled codes of a service as well as related resources required to deploy this service within its corresponding service container. The Service Replica/Instance component deploys the redeployable service package within its corresponding service container. The Monitoring Service component monitors the performance of service/application instances and sends alert if a service instance in underloaded or overloaded. The Service Load Balancer component manages all instances of a service belonged to an application instance. Finally, The Tenant Configuration Files component contains the customization information for each tenant in multi-tenancy architecture that the load balancer can use when creating new application/service instances.

In [32], authors proposed an elasticity approach to ensure the QoS of multi-tier Cloud applications while reducing costs. The proposed approach aims at detecting the bottlenecks tiers in multi-tier applications and accordingly scales up or down these overloaded tiers. To this end, they used an analytical model, based on queuing theory, to capture the cost behavior of applications on every unit of time. This cost behavior is used by an elasticity algorithm, called Cost-Aware Scaling (CAS) algorithm to ensure the elasticity of deployed applications while spending as little cost as possible to meet the required QoS. The CAS algorithm operates at the tiers-level by scaling up and down only the overloaded tiers within applications. To do this, the algorithm monitors and analyzes the incoming requests rate to detect workload changes. If a change occurs, the algorithm triggers capacity estimation to obtain the updated server set to add or remove. Using the result of this estimation, it updates the number of servers by adding or removing servers in order to meet QoS requirements. On one hand, the scaling up algorithm aims at reducing application response time while keeping the deployment cost as low as possible. On the other hand, the scaling-down algorithm aim at reducing the cost as much as possible while maintaining a satisfying response time.

These two approaches discussed and proposed mechanisms for ensuring the elasticity at the service scope. Nevertheless, the correctness of the proposed mechanisms

is not proved since these approaches are not based on formal models. In addition, the proposed mechanisms are not generic since they do not allow the use of different elasticity strategies.

Several works have proposed to use a controller to manage elasticity. In [33], Ali-Eldin et al. focused on ensuring horizontal elasticity of Cloud infrastructure. For this aim, they proposed a hybrid controller that couples a reactive controller for scaling-up and a proactive controller for scaling-down VMs. The reactive controller bases its elasticity decisions on the current load while the proactive controller is basing its decisions on the history of the load. The preference is given for the scaling-up if there is a contradiction between the reactive controller and the predictive one. A Cloud service is modeled, using queuing theory, as a closed loop control system. In this model, the controller has to scale-up or scale-down VMs to ensure that no requests are lost (QoS) while maintaining the number of VMs to a minimum. Authors compared and evaluated, using simulation, different scenarios for coupling reactive and proactive controllers. The evaluation results shown that coupling a reactive and predictive controller is more efficient in maintaining QoS than using only a proactive controller or a reactive controller.

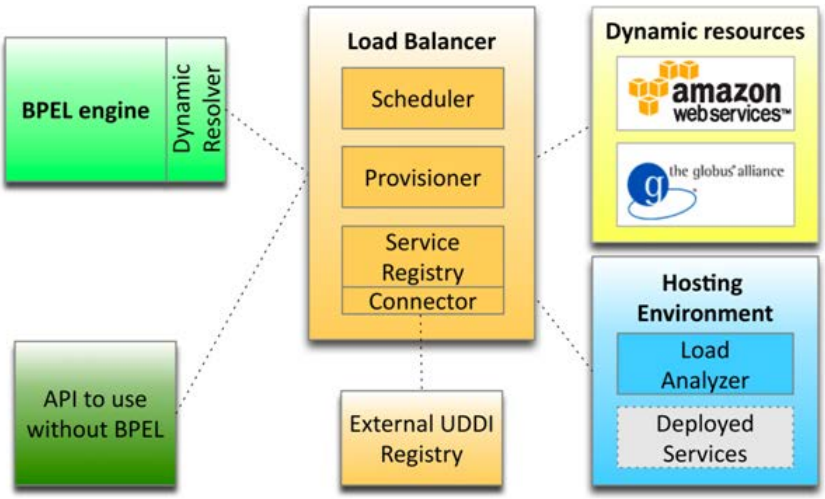


Figure 2.5: Architecture for elastic resource provisioning of BPEL workflows [3].

In [3], authors proposed an approach for elastic resource provisioning of BPEL workflows without changing the BPEL standard. To this end, they proposed to dynamically schedule service calls of a BPEL process over a set of hosts. The scheduling process uses the load of each host as a metric to choose the most appropriate target host. At each step of the workflow execution, the approach schedules the service calls according to the load of the target hosts. When an appropriate host is chosen, the

service call is routed to this host. If all the hosts are overloaded, the provisioning component provides new hosts by starting new virtual machines (in Amazon's EC2 infrastructure) and deploying the required components on them. The overall architecture is presented in Figure 2.5. This architecture is composed of three components: The Dynamic Resolver component extends the BPEL engine's invocation mechanism. The Load Analyzer component collects information about the workload of hosts. The Load Balancer component schedules service calls and manages hosts.

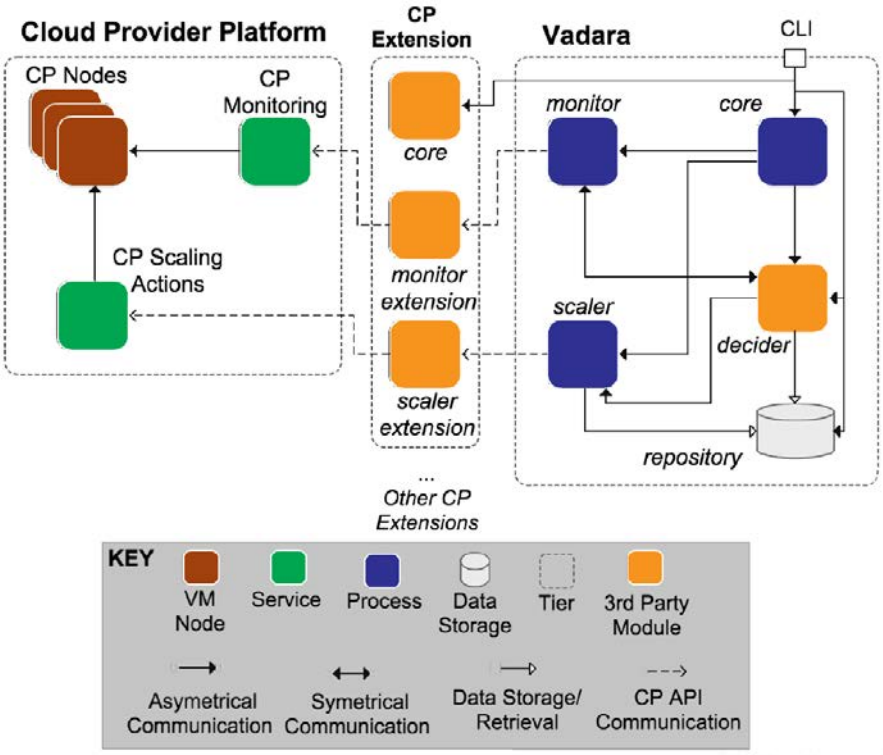


Figure 2.6: Vadara's Architecture for Cloud applications elasticity [4].

In [4], authors proposed Vadara, a generic framework that provides an abstraction layer for existing Cloud providers allowing the use and development of elasticity strategies in a unified manner. The proposed framework allows decoupling elasticity strategies from underlying Cloud platforms, allowing the use of generic strategies. To enforce the elasticity, they proposed to add an extension to Cloud platforms in order to allow the interaction between the Provider's components (i.e., monitoring and scaling components) and the Vadara components. As shown in Figure 2.6, the Vadara framework implements a set of components: a core component that configures and initializes the framework components. A monitor component that collects, aggregates and sends monitoring information to the decider component. The decider

receives monitoring information, analyzes them and sends the appropriate elasticity actions to the scaler component. Afterwards, the scaler component requests the PaaS scaling service in order to execute the elasticity actions chosen by the decider.

In these three latter approaches, the elasticity of Cloud services is managed using an elasticity controller. However the elasticity mechanism (i.e., the controller) serves all the deployed Cloud services and may be not elastic, consequently it could form a single failure point.

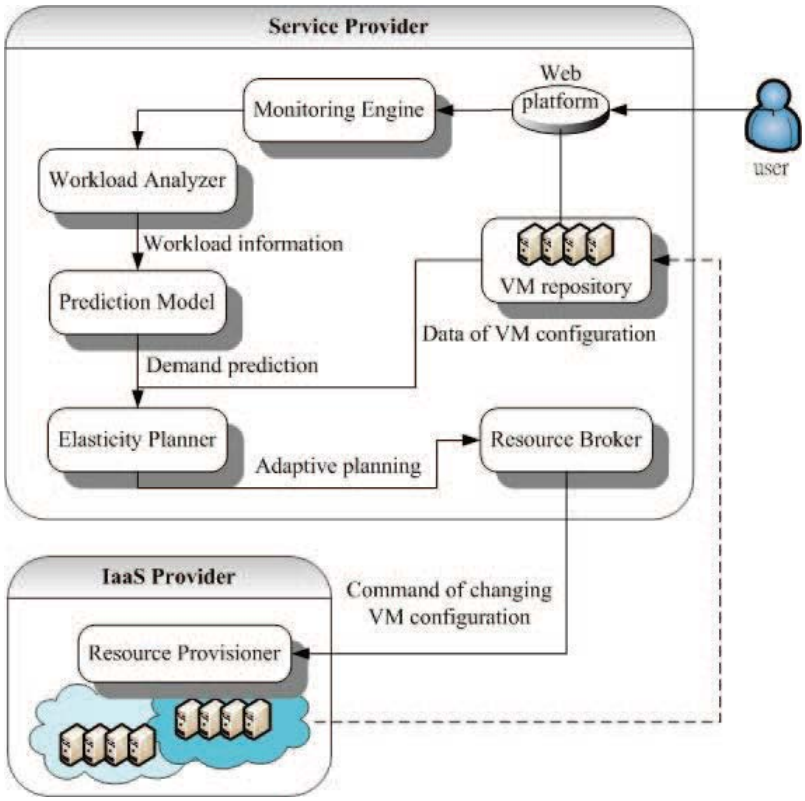


Figure 2.7: Service provider architecture for elastic resource provisioning [5].

The work presented in [5] aims at addressing both vertical and horizontal elasticity as a resource provisioning problem. To this end, authors proposed an algorithm that operates in two phases: The first phase determines the optimal number and types of VMs to be provisioned to respond to the estimated future demand (long-term resource reservation). The second phase acquires more resource if the reserved capacity resource is no longer satisfied (on-demand resource allocation). In addition, a service provider architecture is proposed. This architecture uses the two-phase algorithm in order to ensure elastic resource provisioning while optimizing the operational cost. As shown in Figure 2.7, the proposed architecture is composed of a Monitoring Engine

component that monitors the workload and resource utilization, a Workload Analyzer component that uses analytical models to analyze the workload, a Prediction Model component that predicts incoming demands and an Elasticity Planner component that executes the two-phase algorithm in order to provide an optimal resource provisioning. Finally, a Resource Broker component performs adaptive planning and delivers resource to the IaaS Provider.

In [34], authors proposed an adaptation framework that autonomously scales the amount of servers allocated to an application according to workload variations. The proposed approach aims at reducing energy consumption while maintaining the expected performance. To do this, the framework transforms, using a model-driven approach, design models (applications and system models) into an analyzable stochastic Petri net and generates the adaptation plan. A similar approach was proposed by Ardagna et al. in [35], where they presented a framework for resource management in multitier virtualized systems. This framework aims at ensuring a good tradeoff between performance and energy consumption. To this end, the framework considers both short-term planning (load balancing, capacity allocation and frequency scaling) and long-term planning (server switching and application placement) problems.

Authors of [36], proposed a two-tiered resource allocation mechanism for elastic provisioning of VMs' resources in data centers. The proposed approach consists of local and global resource allocation algorithms based on a two-level control model. The local on-demand resource allocation on each server optimizes, based on scaling threshold values, the resource allocation to VMs (in terms of CPU and memory) within a server in an elastic manner, while the global on-demand resource allocation optimizes indirectly the resource allocation among applications in the entire data center by adjusting the scaling threshold on each local resource allocation. The optimization of resource allocation is done by preferentially giving resource to critical applications when a resource concurrency occurs.

In all these four latter approaches, the elasticity is addressed only at the infrastructure scope (scaling up/down VMs) and remains not sufficient to ensure the elasticity of deployed applications since they do not consider the elasticity at the application/service scope.

### 2.3.3 Evaluation of Elasticity Strategies

To the best of our knowledge, few works were interested in the evaluation of elasticity strategies.

In [37], authors proposed an approach for measuring and evaluating elasticity of different Cloud platforms. They defined workload profiles to simulate different realistic load variation. These workload profiles are defined using a set of mathematical functions (linear and exponential functions, etc.). Particularly, the proposed approach allows Cloud users to measure and evaluate the financial implication of using elasticity facilities offered by different Cloud platforms. The elasticity is measured at the

point of view of users by analyzing resource provisioning behaviors on these Cloud platforms. The measurement approach consists in calculating the financial penalty attributed to user on the occurrence of under-provisioning or over-provisioning of resources. The evaluation of an elasticity strategy is related to its efficiency in enhancing the users' benefits.

In [6], Suleiman et al. proposed an analytical model, using queuing theory, to evaluate the influence of elasticity strategies on the performance of 3-tier applications deployed on Cloud infrastructures. The proposed approach allows cloud users to evaluate different thresholds-based elasticity strategies and choose the most appropriate one for their applications. Particularly, they studied the impact on application' performance (in terms of CPU utilization, application response time and the number of servers) when the threshold values used by an elasticity strategy are changed. The 3-tier applications are modeled using queuing theory (see Figure 2.8). Each server is represented as a queue. The web server acts like a Load Balancer by routing the incoming requests over the set of application servers. Each application server sends queries to the database server. The application tier is modeled as an M/M/m queue where the servers are variable (m) and the calls arrival is represented as a Poisson process. The proposed model can approximate the values of metrics such as CPU utilization and the number of servers needed to ensure the satisfaction of demands. In addition, they proposed scaling out and in algorithms, based on 3-tier applications model, that simulate thresholds-based strategies that use CPU as metric to perform elasticity actions on Cloud infrastructures. Note here, that this study consider only reactive strategies and do not take into account other kind of strategies (e.g., predictive).

In [38], authors proposed new metrics to evaluate the behavior of elasticity. Particularly, they defined the speed metric that consists in the average time it takes to switch from an under-allocation state to a normal or over-allocation state for the scaling up, and in the time to switch from an over-allocation state to a normal or under-allocation state for the scaling down. In addition to the speed metric, a precision metric is proposed. This metric can be considered as the difference of the current amount of allocated resource from the real resource demand. In [39], authors extended this previous work by proposing a benchmark methodology for the evaluation of existing Cloud infrastructures. The evaluation approach compares the evolution of resource demand with the quantity of resource allocated in order to analyze and evaluate the elasticity of different Cloud infrastructures. To do this, they used load variation profiles that consist in a mixture of several patterns. These load profiles allows producing the same resource demand on all compared platforms according to the elasticity characteristics of these infrastructures.

In [7] authors proposed a formal model for quantitative analysis of elasticity at the infrastructure scope. To this end they proposed to use Markov Decision Processes (MDP) to model elasticity actions. On one hand, the abundance of possible elasticity actions is represented as a non-deterministic aspect. On the other hand, the effects

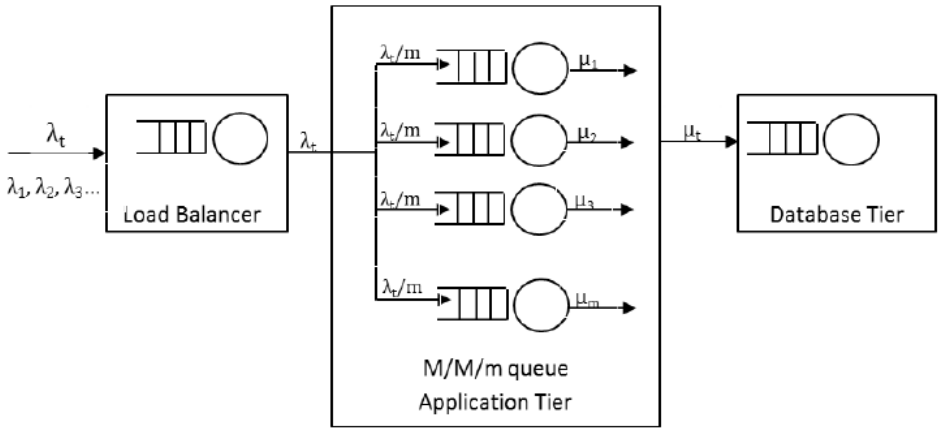


Figure 2.8: The analytical model for elasticity evaluation [6].

of elasticity actions are represented as a probabilistic aspect. As shown in the MDP model of Figure 2.9, each state represents a different cluster size (i.e., the number of VMs). Transitions between states of the model represent elasticity actions (i.e., scaling the number of VMs). A transition starts from the actual state and goes to the state that corresponds to the execution of the elasticity action (e.g., the state after adding a new active VM). These transitions are mapped to a probability that represents the probability to execute a specific elasticity action. Based on MDP models, the proposed approach uses continuous online verification in order to execute elasticity actions. To do this, the approach starts by instantiating a model according to the workload and environment conditions. Then, this model is online verified in order to execute elasticity actions. The proposed approach allows evaluating, in terms of maximizing the system utility (i.e., avoiding over-provisioning), different elasticity models and policies (reactive, Q-learning reinforcement learning etc.).

In [8], authors proposed ADVISE (evAluating clouD serVIce elaSticity bEhavior) which is a framework for the evaluation of Cloud service elasticity behavior. This framework is based on a learning process and a clustering-based evaluation process that determines at runtime the expected elasticity behavior of Cloud service. On one hand, this framework can be used to improve the decision quality of elasticity controllers. On the other hand, it can be used in order to evaluate different elasticity control processes and determine the most appropriate one regarding the considered Cloud service and a particular situation. As shown in Figure 2.10, the Learning process captures at some points of Cloud service execution (just after the enforcement of the elasticity control process) information about the different metrics that can influ-

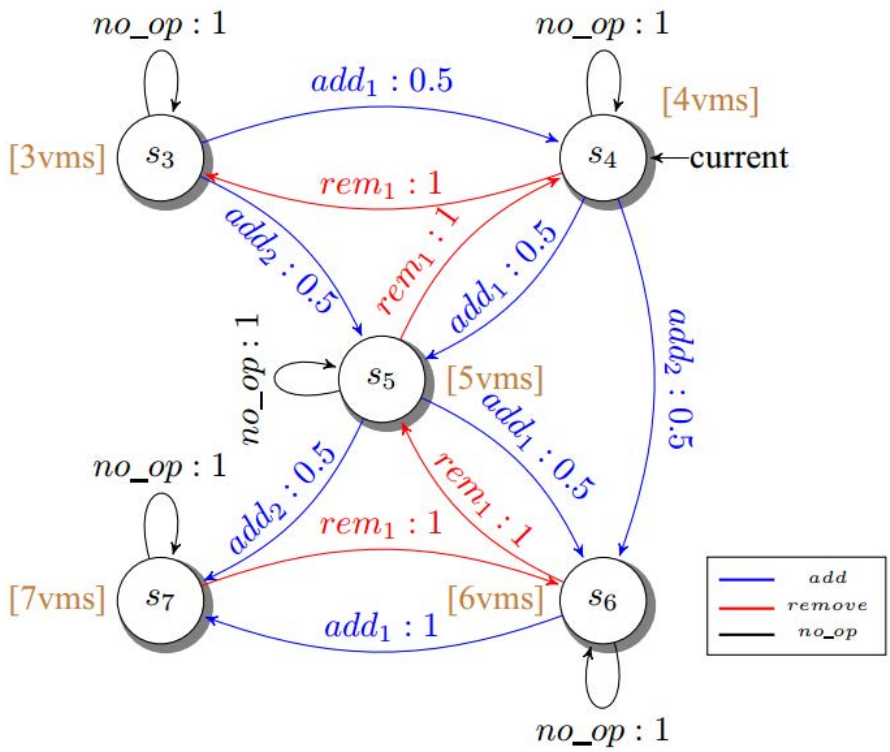


Figure 2.9: Service provider architecture for elastic resource provisioning [7].

ence the behavior of Cloud service such as service structure and workload, deployment strategies, the resource used by the service and the control processes enforced, etc. Then, it transforms these metrics information to multi-dimensional points to compute and evaluate the expected elasticity behavior.

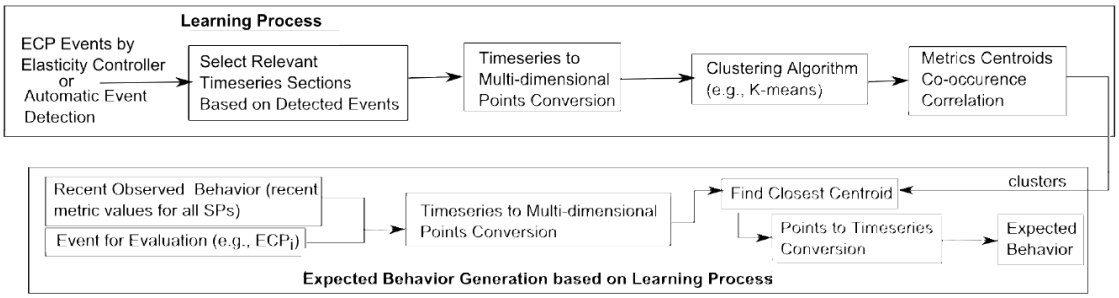


Figure 2.10: Modeling Cloud service behavior process [8].

Almost all of these works use elasticity strategies that are limited to infrastructure metrics (e.g., CPU utilization) to base their decisions and do not consider metrics related to deployed applications. In addition, these latter approaches are limited to simulation-based (performance) evaluations and do not allow formal verification of the correctness of elasticity strategies (e.g., model checking techniques).

### 2.3.4 Provisioning of Elasticity Mechanisms in the Cloud

In [9], the authors introduced the Vienna Platform for Elastic Processes (ViePEP), a platform for realizing elastic processes in Cloud environment. Figure 2.11 presents the architecture of the ViePEP platform. This platform is composed of a set of components for controlling workflows executions (i.e., Workflow Manager component), load balancing between service invocations (i.e., Load Balancer component) and optimizing the process landscape (i.e., Reasoner component). ViePEP allows the monitoring of the process execution and the reasoning about optimizing resources utilization using the current and future system landscape. To do this, the platform can carry out a set of elasticity actions *e.g.,* starting (stopping) a new VM which hosts an overloaded (underloaded) service. In [40, 41], the authors extended ViePEP with two works. The first extension consists on a prediction and reasoning algorithm, based on knowledge about the current and future process landscape, for elastic process execution. The second extension consists on a scheduling and resource allocation algorithms, based on user defined non-functional requirements, in order to optimize resources utilization.

In [10], authors proposed to modify actual Cloud infrastructures in order to support resources elasticity. As shown in Figure 2.12, they proposed to add a Forecasting engine into the core layer of the OpenNebula architecture. This Forecast engine hosts a prediction model that predicts at runtime the expected demand of applications hosted within VMs. To do this, this engine takes as input the user's workload and predicts the optimal resource requirements. The workload is defined using request rate and the number of requests per unit time while the performance is defined using the server processing time. The predicted resource requirements are used by the resource manager of the Cloud infrastructure to provision the needed amount of resource. In addition, they discussed a cost model to fix an optimal trade-off between over-allocation and under-allocation of resource by considering over-allocated resource as cost (paying for unnecessary resources) and under-allocation as a penalty cost (violation of the SLA).

Though these two latter approaches discuss provisioning of elasticity mechanisms in the Cloud, they seem to be difficult to use in real Cloud environments. On one hand, the ViePEP approach requires an effort to replace the existing engines for running elastic processes. On the other hand, the second approach is difficult to use in real world since it requires modifying actual Cloud infrastructures in order to support resources elasticity.

In [11], the authors presented ElaaS, a service for managing elasticity in the Cloud.

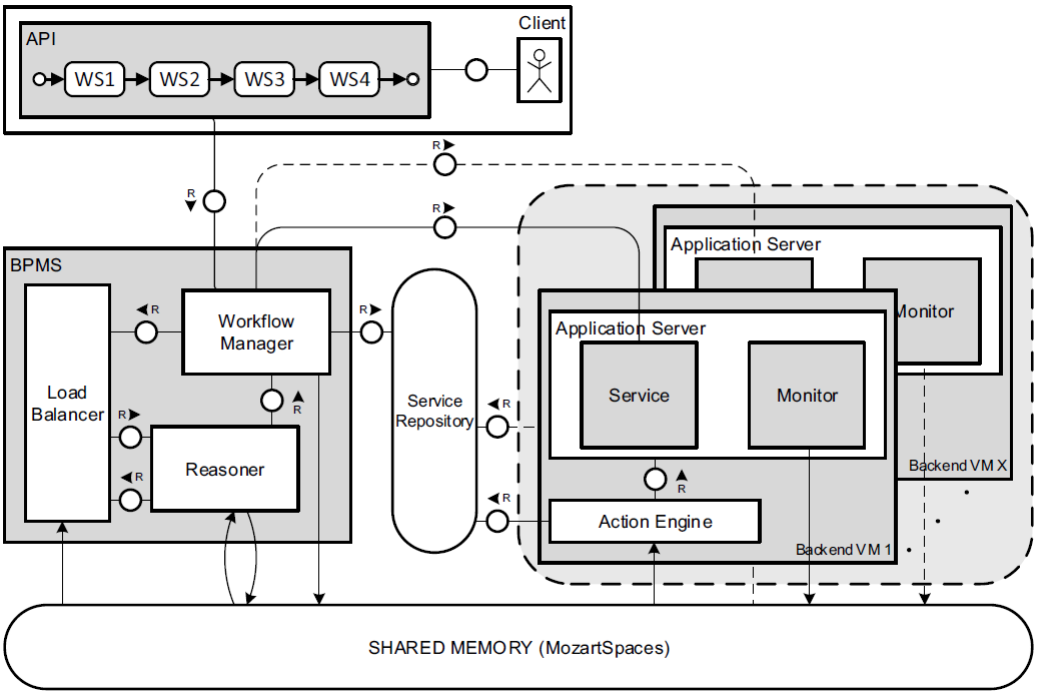


Figure 2.11: ViePEP Architecture [9]

ElaaS is implemented as a SaaS application that can be used in any Cloud environment and Cloud-enabled application. It is composed of a set of pluggable components (Figure 2.13): The ElaaS Core component coordinates the activities and processes. The Application Manager acquires and analyzes information related to the user and the application. The role of the Monitoring Manager is the communication with the monitoring sources that can be at different layers (infrastructure, platform or software specific monitoring KPIs). The Business Logic Manager is the decision making component of the Framework. Note here that the business logic is not included in the component in order to allow the use of different business logic depending on the specific requirements of each application. Finally, The Action Manager relays the action messages to the appropriate components of the application and/or platform and produces a new deployment graph according the executed action. Therefore, application elasticity is insured based on the deployment graph of the considered application and its KPI.

In [42], the authors presented a framework for modeling and reasoning about non-functional properties that should reflect elasticity of deployed business services. The proposed framework considers not only elasticity properties but also business objectives and constraints in order to find possible service provisioning solutions. To do

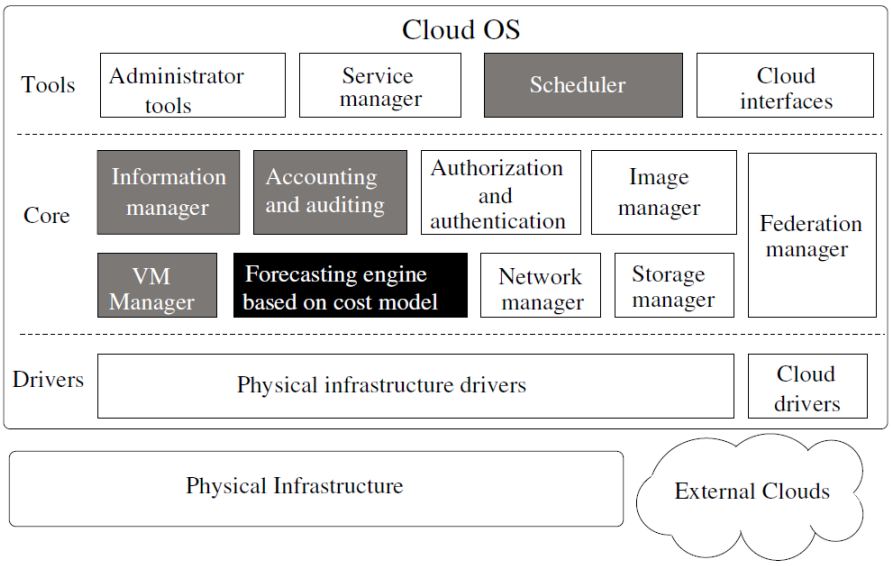


Figure 2.12: Extending the OpenNebula architecture with a Forecasting engine [10].

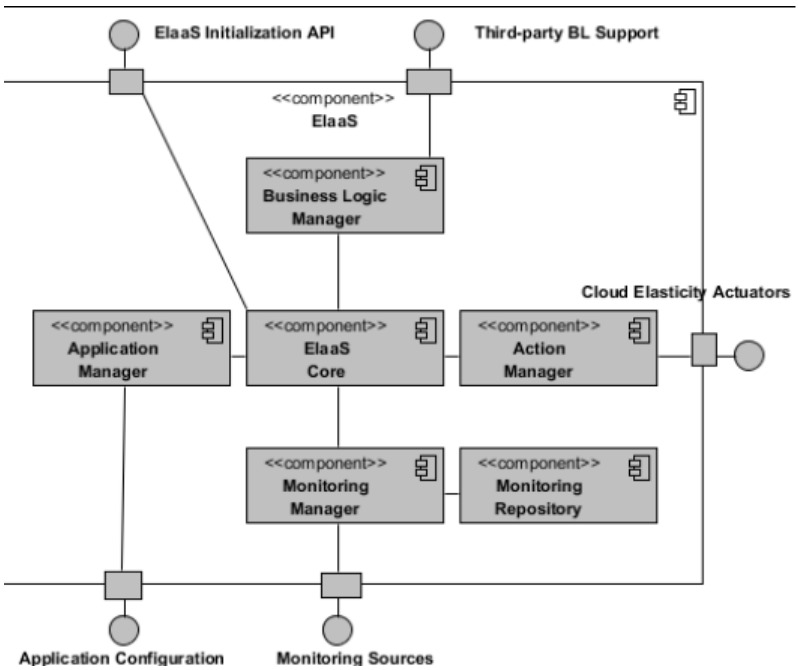


Figure 2.13: ElaaS Components [11]

this, the framework translates business objectives into a set of logic predicates and optimization objectives. Business constraints are represented as a fact while elasticity reasoning mechanisms are represented as a set of rules. As defined the elastic properties and mechanisms can tackle any application, since the characteristics of business processes (structure or behavior) are not considered in the proposed approach.

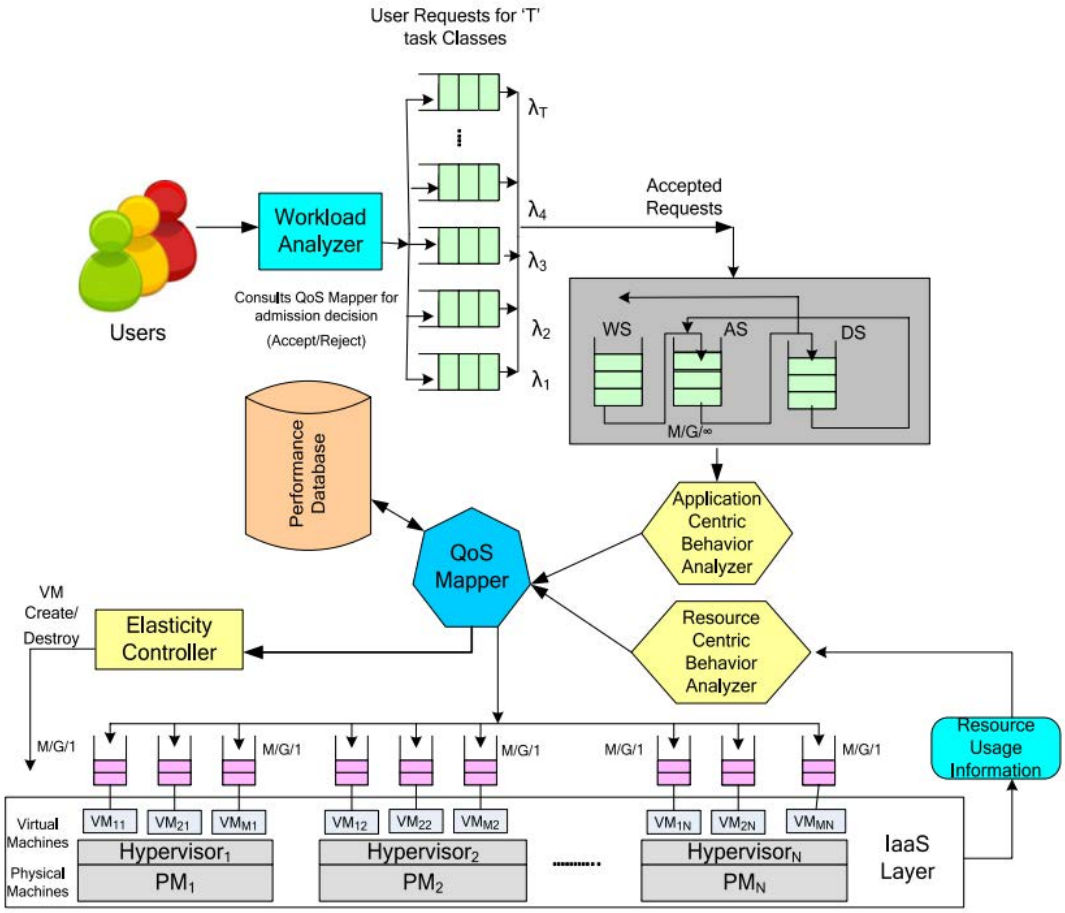


Figure 2.14: The QoS-Aware Resource Elasticity (QRE) framework [12]

Authors of [12], proposed the QoS-Aware Resource Elasticity (QRE) framework for ensuring resource elasticity of application while considering QoS requirements. The proposed approach maps the application specific QoS attributes (in term of response time and resource utilization) to Cloud resource specific attributes in order to ensure QoS-aware resource elasticity. To this end, the proposed framework captures, using an analytically model, the application behavior in order to estimate the application workload needs in terms of allocated resources so that the QoS requirements are

always satisfied. As shown in Figure 2.14, this framework is composed of the Workload Analyzer component that interacts with the applications users, and consults the QoS mapper component to decide if a request is accepted or not. The Application Centric Behavior Analyzer component analyzes the behavior of application in order to predicate the arrival rate of different tasks with respect to the variation of arrival rate and workload. The Resource Centric Behavior Analyzer component obtains, using the IaaS monitoring component, the current resource usage information. The QoS Mapper component maps the application specific QoS requirements to resource specific allocations. To do this, it checks the Performance Database that contains the workload mix and QoS attributes. Finally, the Elasticity controller component receives the requests from the QoS Mapper for adding or removing VMs to the Cloud infrastructure.

Authors of [13, 43] proposed a generic framework for managing elasticity of Cloud applications. To this end, they proposed a Planification component integrated into a MAPE (Monitor, Analyze, Plan and Execute) loop to provide elasticity while considering complex elasticity scenarios. The Planification component is about how an application must be reconfigured according to an elasticity decision. As described in Figure 2.15 When the Planification component receives an elasticity decision from the Analyzer component, the Planification computes the new state of the considered application (i.e., the new application architecture). To do this, it uses the initial Extensional Model that describes the current state of the considered application (i.e., the current application architecture) and the Intensional Model that consists in a template for all possible Extensional Models for the considered application (i.e., all possible architectures of the application). The algorithm uses these two models to compute and determine how the application will be modified (i.e., the new application architecture) according to the elasticity decision (final extensional model). These modifications consists in adding/removing a component/container, binding/unbinding components, placing a component/container into a container, setting/unsetting a parameter of a component/container.

While the idea of pushing elasticity management to the applications is interesting, these four latter approaches are difficult to use since they assume an effort from the application designer/administrator who will be in charge of delivering necessary information for elasticity enforcement.

### 2.3.5 Synthesis of Related Work

In the literature, almost of the presented works dealing with the elasticity focus on the infrastructure scope [5, 34, 35, 36]. These approaches allow the elasticity of VMs (i.e., adding/removing VMs according to workload variation) but they are not sufficient to ensure the elasticity of deployed applications since they do not take into account the nature of the application e.g., service-based applications.

To the best of our knowledge, most of the existing proposals related to elasticity

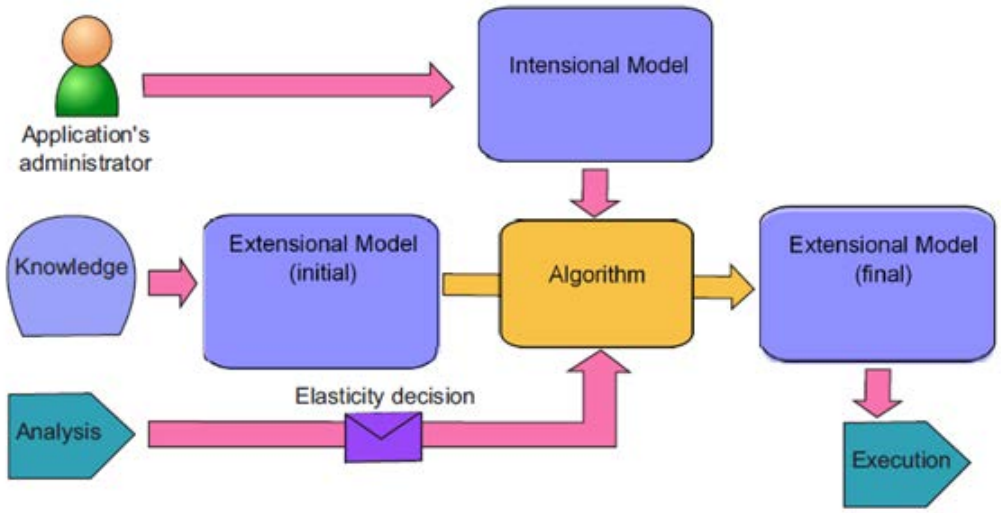


Figure 2.15: Planification component for the elasticity of Cloud applications [13]

at the application scope mainly those we cite above are not suitable for service-based applications. We think that addressing elasticity at the application scope is not sufficient for ensuring the elasticity of this kind of applications. Elasticity should also be ensured at the scope of services that compose these applications. In fact, we argue in our work that it is unnecessary to duplicate or consolidate the entire application and consequently all the services that compose it while the bottleneck can come from one or some services of the application. In this context, some works have been proposed to address the elasticity at the scope of services [2, 32]. Nevertheless, the correctness of the proposed mechanisms is not proved since they are not based on a formal model.

Several approach proposed to use a controller to manage Cloud services elasticity [33, 3]. However, the controller manages all the deployed Cloud services, which can leads to the overload of the controller and consequently it could form a single failure point. It would be interesting to be able to assign an elasticity controller at different granularities according to environment requirements: a single controller for all the deployed processes, a controller for each tenant (that corresponds to an enterprise) or even a controller for each deployed process.

Another critical point in addressing elasticity is the ability to evaluate elasticity strategies. In fact, it is necessary to be able to evaluate these strategies to guarantee their effectiveness before using them in real Cloud environments. To the best of our knowledge, none of the existing works proposes a formal approach for the verification of the correctness of elasticity strategies. Indeed, almost approaches are limited to

simulation-based (performance) evaluations [37, 6, 38, 39, 7, 8] and do not allow formal verification of the correctness of elasticity strategies (e.g., model checking techniques).

There are two main approaches for the provisioning of elastic processes in Cloud environments. The first approach consists in defining a novel environment for the execution of elastic processes [9, 40, 41, 10]. Nevertheless, this kind of approach seems to be difficult to use in real world since it requires an effort to replace/modify the existing engines and/or Cloud infrastructures. The second approach that we propose to adopt in our work consists in providing mechanisms that allow provisioning of elastic processes without changing the nature of the SBPs or existing Cloud environments.

In our work, we propose an elasticity approach that responds to all the discussed criteria. Accordingly, in chapter 3, we propose an elasticity approach that intertwines a formal model for SBPs elasticity that operate at the scope of services with a generic controller for elasticity that can be used with different elasticity strategies and at different granularities. In chapter 4, we propose an approach for evaluating SBPs elasticity using both verification-based and simulation-based approaches. In chapter 5, we propose two approaches for the provisioning of elastic SBPs in real Cloud environments (IaaS and PaaS contexts).

## 2.4 Conclusion

In the first part of this chapter, we introduced the basic concepts related to our work. We presented Cloud Computing as the target environment of our research. Afterwards, we presented the Service Oriented Architecture and a specified type of application that we target namely Service-based Business Processes (SBPs). Then, we presented the principle of elasticity.

In the second part of this chapter, we proposed an overview of the existing work on different aspects related to elasticity. To this end, we started by discussing proposed models and mechanisms for elasticity. Then, we were interested in proposed approaches for the evaluation of elasticity. Afterwards, we studied works that deal with provisioning of elasticity mechanisms in Cloud environments. Finally, we draw a synthesis and we discuss how these works respond or not to the criteria listed in 2.3.1.

Chapter 3

# Formal Model for SBPs Elasticity

## Contents


**3.1 Introduction** . . . . . **45**
**3.2 Modeling Elasticity of SBPs** . . . . . **46**
3.2.1 Illustrating Example . . . . . 46
3.2.2 Elasticity Mechanisms . . . . . 47
3.2.3 Modeling Requirements . . . . . 48
**3.3 Modeling of Stateless SBPs Elasticity** . . . . . **50**
3.3.1 Stateless SBP Modeling . . . . . 50
3.3.2 Elasticity Operations . . . . . 52
3.3.3 Correctness of Elasticity Operations . . . . . 55
**3.4 Modeling of Stateful SBPs Elasticity** . . . . . **58**
3.4.1 Stateful SBP Modeling . . . . . 58
3.4.2 Elasticity Operations . . . . . 61
**3.5 Modeling of Timed SBPs Elasticity** . . . . . **63**
3.5.1 Timed SBP Modeling . . . . . 63
3.5.2 Elasticity Operations . . . . . 65
**3.6 Generic Controller for SBPs Elasticity** . . . . . **68**
3.6.1 Elasticity Controller . . . . . 68
3.6.2 Formal Description of the Generic Controller . . . . . 70
3.6.3 Illustrating Example of SBPs elasticity . . . . . 72
**3.7 Conclusion** . . . . . **75**


## 3.1 Introduction

SBPs elasticity can be ensured by providing Cloud environments with elasticity mechanisms so that they can be able to adapt to the workload changes while ensuring the desired functional and non-functional properties. Therefore, it is necessary to define

a model and mechanisms that allow describing and ensuring SBPs elasticity. The elasticity aims at ensuring the QoS of SBPs despite workload fluctuation. The QoS can be impacted by the variation of the workload (e.g., in terms of number of invocations) and measured using many indicators (response time, availability, reliability, etc.). Therefore, it is necessary to have a model that allows representing the SBP workload while allowing the measure of elasticity indicators (e.g., response time, resources consumption). We argue, in our work, that it is beneficial to adopt formal models to describe SBPs elasticity which provides rigorous description and allows verification of properties.

In this chapter, we present our proposal to formally model SBPs elasticity. We start by introducing our approach for modeling elasticity of SBPs. In our approach, we advocate that handling elasticity does not only operate at the process scope but it should also operate at the scope of services. Therefore, we propose a model for describing SBPs while considering the notion of services workloads and services response times. Using this SBP model, we will be able to define elasticity operations (duplication/consolidation) that operate at the scope of services to ensure the elasticity of stateless [44], stateful [45] and timed [46] SBPs. In our work, SBPs are modeled as Petri nets and elasticity operations (service duplication/consolidation) are defined and their correctness is formally proved [47]. Further, we will show how we can intertwine our elasticity operations (duplication/consolidation) with a controller to ensure SBPs elasticity. For this, we define, using high level Petri nets, an elasticity controller that monitors SBP execution, analyzes monitoring information and executes the appropriate action (service duplication/consolidation) in order to enforce the elasticity of deployed SBPs. This controller is generic in order to allow the implementation and execution of different elasticity strategies.

## 3.2 Modeling Elasticity of SBPs

In this section, we introduce our approach for modeling elasticity of SBPs. To this end, we start by illustrating our approach with a motivating example of an online computer shopping SBP. Then, we propose elasticity mechanisms for ensuring SBPs elasticity at the scope of services. Afterward, we discuss the requirements for modeling SBPs.

### 3.2.1 Illustrating Example

Herein, we present an example of an online computer shopping SBP composed of four services. This example will be used for illustration along this dissertation. Figure 3.1 presents the BPMN [48] model of the considered SBP. The four services that compose the SBP are:

- **Requests service ($S1$):** receives requests to purchase a computer. Once the service receives a request, it calls the Computer assembly service and the Invoice

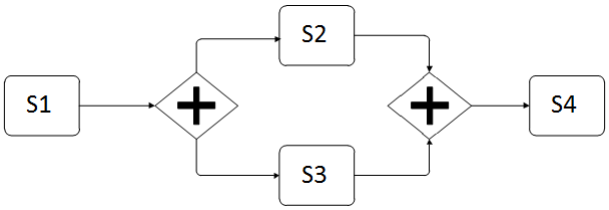


Figure 3.1: BPMN model of the SBP example of the online computer shopping service.

service.

- **Computer assembly service ($S2$):** performs the assembly of computer components according to the requester desires.
- **Invoice service ($S3$):** makes the invoice related to the purchased computer.
- **Delivery service ($S4$):** delivers the computer with its invoice to the customer.

In our example, assuming that at a certain time the load of the SBP overcomes its maximum capacity which can leads to loss of QoS, we have to make sure that the SBP still runs and that all the requests are processed while maintaining the desired QoS. One solution to maintain the SBP QoS would be to duplicate as many times as necessary the entire process (and so, all the services that compose the SBP). However, this solution creates an over-provisioning of resources caused by the unnecessary duplication of non-overloaded services. Therefore, we think that duplicating only the bottleneck services is a better alternative while avoiding unnecessary resources consumption. In 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 to maintain the QoS of the entire SBP. So, all we have to do is to duplicate the overloaded service ($S2$ in this case). Duplicating this service consists on adding as copy as needed to handle the incoming load while maintaining the required QoS.

Furthermore, assuming that at a certain time the load of the requests service decreases whereby the service and its copies use more resources than required for the same QoS. In this case, we have to consolidate the unnecessary copies of the service in order to avoid resources under-utilization and thereby optimizing consumption of Cloud resources.

### 3.2.2 Elasticity Mechanisms

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 services into a SBP can be ensured using any appropriate service composition specifications (*e.g.,* BPMN [48] or BPEL [49]).

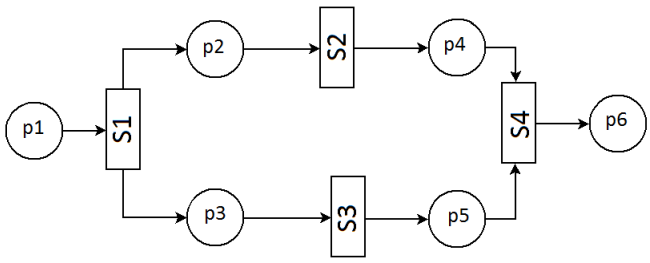


Figure 3.2: SBP execution model of the online shopping service

SBPs elasticity can be ensured by providing Cloud environments with mechanisms that allow a deployed SBP to scale up or down whenever needed. To scale up a SBP, these elasticity mechanisms must duplicate the process in order to create as instance as needed to handle the dynamically received requests. To scale down a SBP, these mechanisms should consolidate the process in order to remove useless instances, thereby avoiding under-utilization of resources.

We believe that elasticity mechanisms (duplication/consolidation) do not only operate at the process scope but they should also operate at the scope of services. In fact, we argue that it is unnecessary to duplicate or consolidate the entire SBP and consequently all the services that compose it while the bottleneck can come from one or some services of the SBP.

In order to manage SBPs elasticity using duplication and consolidation operations, an elasticity controller can be used. This controller monitors SBP execution, analyzes monitoring information and executes appropriate actions (duplication/consolidation) in order to ensure elasticity of deployed SBPs.

### 3.2.3 Modeling Requirements

To model SBPs, several techniques can be used (BPEL [49], BPMN [48], Petri nets [50], etc.). In our work, we are interested in the formal aspect of modeling. So, we model the SBP using Petri nets. Many approaches have been used to model SBPs using Petri nets. Generally these approaches represent the SBP execution model which specifies how the processes and their services need to be executed and in what order. In this latter, each service is represented by a transition. The places represent the states between services. The execution model of the SBP of Figure 3.1 gives the Petri net shown in Figure 3.2. As shown in this Figure, SBP services are represented by transitions ($S1$, $S2$, $S3$ and $S4$) while the states between services are represented by places ($p1$, $p2$, $p3$, $p4$, $p5$ and $p6$).

The SBPs execution model is suitable to verify behavioral properties. Nevertheless, this model does not provide a view of the evolution of loads on services which is necessary to verify non-functional properties such as elasticity. Therefore, it is inter-

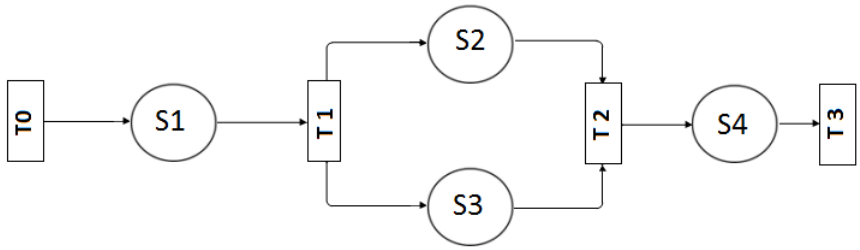


Figure 3.3: Petri net of the SBP of the online computer shopping service.

esting to have a view of the way services are deployed and their loads. For that reason, we propose to model SBPs using their deployment model in which we can represent the way a process and its services are deployed and the load on each service of the SBP. In our 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. The SBP deployment model of the SBP execution model of Figure 3.2 gives the Petri net shown in Figure 3.3. As shown in this Figure, SBP services are represented by places ($S1$, $S2$, $S3$ and $S4$) while the calls transfers between services are represented by transitions ($T0$, $T1$, $T2$ and $T3$). The invocation of the SBP starts with the execution of the service $S1$. The invocation of a service consists in adding a token to its corresponding place (e.g., executing the service $S1$ consists in adding a token to the place $S1$). Transferring a call from a service (or a set of services) to another service (or other services) consists in firing the transition that connects these services (e.g., transferring a call from the service $S1$ to services $S2$ and $S3$ consists in firing the transition $T1$ which removes a token from the place $S1$ and adds a token in both places $S2$ and $S3$). An invocation (a call) is processed when its corresponding token is consumed by the firing of the transition $T3$.

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 not expressible on the execution model. This allows 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.

To model SBPs, we have also to consider the different kind of SBPs. In fact, services involved in a SBP can be stateless or stateful services. On one hand, 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. The different events (or some of them) and interactions that occurred during service execution are taken into account to manage the service invocations. The state of a stateful service can be represented by the user's sessions and the data values specific to this service. In addition, SBPs can be provided with temporal constraints on the control flow of the SBPs. These timed SBPs allow having information about temporal properties (delays, deadlines, etc.) on the SBPs execution. It is obvious that we have to consider these kind of SBPs in our modeling approach of SBPs elasticity.

In this section, we presented our approach for modeling elasticity of SBPs. In our approach, we argued that the elasticity must be ensured at the scope of services. In the following, we propose a formal model for stateless SBPs elasticity (Section 3.3). Afterward, we will propose two extensions to consider stateful SBPs (Section 3.4) and timed SBPs (Section 3.5).

## 3.3 Modeling of Stateless SBPs Elasticity

In order to ensure elasticity of stateless SBPs, we start by introducing our formal model, based on Petri nets, to describe SBPs and two elasticity operations (duplication and consolidation) that allow adding and removing copies of services while preserving the semantics of the SBP. In addition, we prove the correctness of the proposed elasticity operations.

### 3.3.1 Stateless SBP Modeling

To model stateless SBPs we used place/transition Petri nets (PN) [50]. 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 3.1 gives the Petri net shown in Figure 3.3.

**Definition** A stateless SBP 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 $T$. We ignore the $_{\equiv_T}$ and $_{\equiv_P}$ if it is clear from the context.

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 $P' \subseteq P$, $[P']_{\equiv_P} = \{[p]_{\equiv_P} | p \in P'\}$.

**Definition** Let $N$ be a Petri net, we define a net system $S = \langle N, M \rangle$ with $M : P \to \mathbb{N}$ a marking that associates to each place an amount of tokens. The marking is also extended to equivalent classes *i.e.,* $M([p]) = \sum_{p' \in [p]} M(p')$.

In our model, the marking of a place models the number of invocations on its corresponding service (i.e., each token represents a service call). The number of invocations on a service can be considered as the workload of this service.

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. We assume in our model that all service calls are treated in the same manner.

**Definition** 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** 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 $M' \in \{M'' | \exists t' \in [t] : M[t'\rangle M''\}$.

### 3.3.2 Elasticity Operations

Elasticity operations are mechanisms to scale-up and down SBPs when needed. On one hand, 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 (create a new instance of this service) in order to increase service capacity and ensure the desired QoS with respect to the increased load.

On the other hand, when a service has few calls, it will use more resources than required for the same QoS. A solution to this issue is to consolidate the service (remove an unnecessary instance of this service) in order to avoid under utilization of resources.

Hereafter, we give the definition of two elasticity operators that duplicates/consolidates a service.

#### Place Duplication

**Definition** Let $S = \langle N, M \rangle$ be a net system and let $p \in P$, the duplication of $p$ in $S$ that creates a new place $p^c$ ($\notin P$), noted as $D(S, p, p^c)$, leads to 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 $\equiv_{T'} = \equiv_T \cup \{(t, t^c) | t^c \in T'' \wedge t^c = n^{-1}(t)\}$. 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:

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

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

**Place Consolidation**

**Definition** 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 $(p^c)^\bullet \cup^\bullet p^c$
- $M' : P' \to \mathbb{N}$ 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.

**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 3.3.1** *Let $N$ be a well-defined net, the following properties are held on $N$:*

$$\forall t_1, t_2 \in T : [t_1] = [t_2] \Rightarrow [{}^\bullet t_1] = [{}^\bullet t_2] \wedge [t_1^\bullet] = [t_2^\bullet] \tag{3.1}$$

$$\forall p_1, p_2 \in P, t \in T : p_1, p_2 \in ({}^\bullet t \cup t^\bullet)$$

$$\Rightarrow p_1 = p_2 \vee [p_1] \cap [p_2] = \emptyset \tag{3.2}$$

$$\forall t \in T : |[t]| = \prod_{p \in ({}^\bullet t \cup t^\bullet)} |[p]| \tag{3.3}$$

**Proof** The proof of equations 3.1 and 3.2 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 3.3 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 3.3 is true for $S$ *i.e.,*

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

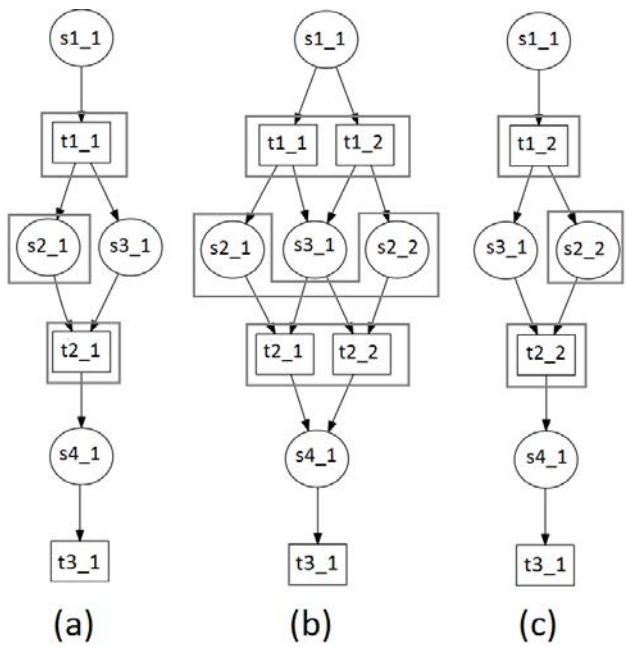


Figure 3.4: Example of the elasticity of a SBP

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:

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

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

**Example** Figure 3.4-(a) shows an example of nets system that represents the SBP of the computer shopping service described previously. The relations $\equiv_P$ and $\equiv_T$ are the identity relations. Figure 3.4-(b) is the resulted system from the duplication of $s2_1$, $D((a), s2_1, s2_2)$. Figure 3.4(c) is the consolidation of the place $s2_1$ in its copy $s2_2$, $C((b), s2_2, s2_1)$. The boxes represent the equivalence relations.

### 3.3.3 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.

**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 $\rho$ as the union of the two functions. Two net systems $S$ and $S'$ are equivalent according to $\rho$, 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 3.3.2** *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 (3.1) to (3.3) we can conclude the proof of the proposition.

**Proposition 3.3.3** *Let $S = \langle N, M\rangle$ and $S' = \langle N', M'\rangle = D(S, p_1, p_1^c)$. Let $\rho$ 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 $M_1$:
$(\Leftrightarrow)\forall p \in^\bullet t : M_1(p) \geq 1$
$(\Leftrightarrow)\forall p \in^\bullet t : M_1([p]_{\equiv_P}) \geq 1$
$(\Leftrightarrow)\forall p \in^\bullet t : M_1'(\rho([p]_{\equiv_P})) \geq 1$
$(\Leftrightarrow)\forall p \in^\bullet t : \exists p' \in \rho([p]_{\equiv_P}) : M_1'(p') \geq 1$
according to proposition 3.3.2
$(\Leftrightarrow)\exists t' \in [t]_{\equiv_{T'}} : M_1'[t'\rangle$
$(\Leftrightarrow)\exists t' \in \rho([t]_{\equiv_T}) : M_1'[t'\rangle$

**Theorem 3.3.4** *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:*

$$S \sim_\rho S' \text{ where } \rho \text{ is defined as in proposition 3.3.3.}$$

**Proof** First, by definition, $\rho$ is a bijection.

**(a)** By definition of duplication, we have $M'(p_1^c) = 0$, so:

$$\forall p \in P : M([p]_{\equiv_P}) = M'(\rho([p]_{\equiv_P})) \tag{3.4}$$

**(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 $\rho$ (and vice versa).
Let $M_1[t_1\rangle M_2$, according to proposition 3.3.3 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:

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

$$M_1([p']_{\equiv_{P'}}) + (Post'(t', p') - Pre'(p', t')) \tag{3.5}$$

We know also, by definition of the duplication:

$$\forall p \in P, p' \in P', t \in T, t' \in T' : [p]_{\equiv_P} \subseteq [p']_{\equiv_{P'}} \wedge [t]_{\equiv_T} \subseteq [t']_{\equiv_{T'}} \Rightarrow$$

$$Pre(p, t) = Pre'(p', t') \wedge Post(t, p) = Post'(t', p') \tag{3.6}$$

From equations (3.5) and (3.6) we conclude:

$$\forall p \in P : M_2([p]_{\equiv_P}) = M_2'([p]_{\equiv_{P'}}) \tag{3.7}$$

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:

$$\langle N, M_2 \rangle \sim_\rho \langle N', M_2' \rangle \tag{3.8}$$

and By induction on equations (3.4) and (3.8) we conclude that $S \sim_\rho S'$

**Theorem 3.3.5** *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:*

$$S \sim_\rho S' \textit{ where } \rho \textit{ is defined as in proposition 3.3.3.}$$

**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 3.3.4 and 3.3.5 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.

**Remark** Let we consider a net system $S = \langle N, M \rangle$ s.t $\forall p \in P, t \in T : |[p]| = |[t]| = 1$. Let $v$ be a vector of natural number $v \in \mathbb{N}^P$ such that $v^{\mathtt{T}} * \mathcal{C} = \overline{0}$ (i.e $v$ is $p - flot$ of $N$). This means that $\sum_{p \in P} v(p) * M(p) = 0$.
The theorem 3.3.5 proofs also that the vector $v' \in \mathbb{N}^{P'}$ with $\forall p \in P, \forall p' \in [p], v(p) = v'(p')$ is also a $p - flot$ for N' (i.e $v'^{\mathtt{T}} * \mathcal{C}' = \overline{0}$). Identically for $T - flot$.

## 3.4 Modeling of Stateful SBPs Elasticity

As shown in the previous section, 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. However, performing elasticity on stateful service needs more attention. In fact, in a duplication/consolidation approach it is necessary to ensure that the state of the stateful service is taken into account by the elasticity mechanisms at each duplication or consolidation. To solve this problem, we propose to model stateful SBPs using Colored Petri Nets (CPN) [51]. 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 persistence 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 we assume the existence of an elastic database (e.g., NoSQL databases) in which the state of stateful services are stored. Consequently, there is a need to render the process/services elastic but not the database service. In the following, we do not discuss the elasticity of databases since it is out of scope of our work.

### 3.4.1 Stateful SBP Modeling

To model stateful SBP we use Colored Petri Nets (CPN). Place/transition Petri nets do 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 $\Pi$, we range over by using $\pi$. 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 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** (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 $e_i \in E$)[1]. 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** (CPN graph) : A stateful SBP model is a Colored Petri Net graph (CPN graph) $N$=$\langle\Sigma$,$P$, $T$,$cd$,$Pre$,$Post$,$\equiv_P$,$\equiv_T\rangle$, where:

- $\Sigma$ 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 $Pre$ : $P \times T \rightarrow \mathcal{E}$ (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 $s \in P$:

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

- $P = P \cup \{sdb\}$ ($sdb$: 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 $t$.

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** (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 of the variables which are in the input arcs expressions. To each CPN graph, we associate its terms incidence Matrix $C$ ($P \times T \rightarrow \mathcal{M}(\mathcal{E})$) with $C = Post - Pre$.

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

**Definition** (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. We assume in our model that all service calls are treated in the same manner.

**Definition** (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 $u : Var(Pre(., t)) \rightarrow \Sigma$ with $M \geq Pre(., t)^u$ to denote a binding of the input arcs variables. [2]

**Definition** 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 $\exists t' \in [t] : M[t'\rangle^u$

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

**Definition** 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.

### 3.4.2 Elasticity Operations

Herein, we extend our elasticity operations (duplication/consolidation) to consider stateful SBPs.

#### Place Duplication

**Definition** 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. $Post'$): $P' \times T' \to \mathcal{E}$ (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:

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

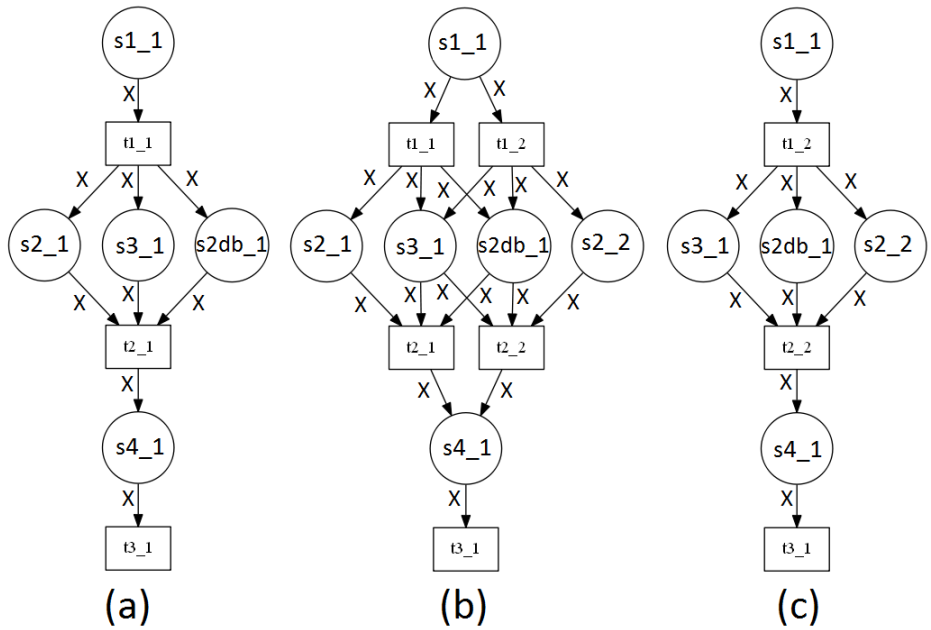


Figure 3.5: An example of the elasticity of the stateful SBP of the online computer shopping service.

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

### Place Consolidation

**Definition** 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 $(p^c)^\bullet \cup^\bullet p^c$
- $M' : P' \rightarrow \mathcal{M}(cd(p))$ with $M'(p) = M(p) + M(p^c)$ and $M'(p') = M(p')$ if $p' \neq p$.

**Example** Figure 3.5-(a) represents the stateful SBP model (empty marking) of the online computer shopping service of Figure 3.1. In this SBP, $s2_1$ is a stateful service and all others are stateless services. $s2db_1$ is the database service that model the data values related to the stateful service $s2_1$. Figure 3.5-(b) is the resulting system from the duplication of the service $s2_1$ in (a), $D((a), s2_1, s2_2)$. Figure 3.5-(c) is the consolidation of the service $s2_1$ in its copy $s2_2$, $C((b), s2_2, s2_1)$.

**Correctness of Elasticity Operations**

In the previous section we applied the same structural duplication and consolidation operations on classical Petri-net. The correctness of duplication/consolidation operators depends on the fact that cardinality of arcs (domain of *pre* and *post* functions) are one or zero. If we restrict in the colored Petri nets the multi-sets expressions on the arcs to a cardinality of one, then the proofs provided in the previous section still holds for stateful SBPs. Note that this hypothesis is very realistic since different instances of SBPs are not supposed to interact with each others. This means that the duplication/consolidation operators preserve the semantics of stateful SBPs.

## 3.5 Modeling of Timed SBPs Elasticity

The two previously introduced models do not allow having information about temporal properties on the SBP execution. This can result in misinterpreting the SBP elasticity. To resolve this, we propose to go further by considering temporal constraints in modeling SBPs elasticity.

### 3.5.1 Timed SBP Modeling

To model timed SBPs, we used Timed Petri Nets (TdPN) [52]. Place/transition and Colored Petri nets do not allow the modeling of time. TdPN have been proposed to extend Petri nets by modeling time information. TdPN are Petri nets where tokens are annotated with a real value that represents the age of the tokens in their current places, arcs connecting places to transitions are annotated with time intervals that represent the values that the age of tokens must have to be consumed by a transition firing.

In our model, each service is represented by a place. The transitions represent call transfers between services according to the behavior specification of the SBP. Tokens represent the calls (invocations) of services. Ages of tokens represent the duration of calls treatment in their current services. In our model, 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.

**Definition** (TdPN) A timed SBP model is a Timed Petri Net (TdPN) $N =< P, T, Pre, Post, \equiv_P, \equiv_T >$:

- $P$: a set of places (represents the set of services/activities involved in a SBP).
- $T$: a set of transitions (represents the call transfers between services in a SBP).
- $Pre \subseteq P \times I \times T$: a finite set of input arcs where $I$ is the set of time intervals defined by $I ::= [a, a] | [a, b] | [a, \infty)$ where $a, b \in \mathbb{N}$ and $a < b$.

- $Post \subseteq T \times P$ : a finite set of output arcs.
- $\equiv_P \subseteq P \times P$: 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: $[t]_{\equiv_T} = \{t' | (t, t') \in \equiv_T\}$.

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

- $p^\bullet = \{t \in T | (p, I, t) \in Pre\}$
- ${}^\bullet p = \{t \in T | (t, p) \in Post\}$
- $t^\bullet = \{p \in P | (t, p) \in Post\}$
- ${}^\bullet t = \{p \in P | (p, I, t) \in Pre\}$

In the following, we define the behaviors (the dynamics) of a TdPN system.

**Definition** (TdPN Marking) Let $N$ be a Timed Petri Net. A marking $M$ on $N$ is a function $M : P \to \mathcal{B}(\mathbb{R}_{>=0})$ that associates to each place a finite multiset of non-negative real numbers that represent the age of tokens that are currently at a given place. The set of all markings over $N$ is denoted by $M(N)$.

**Definition** (TdPN system) A Timed Petri Net system (TdPN system) is a pair $S = \langle N, M \rangle$ where $N$ is a TdPN and M a marking. The marking of a TdPN represents a distribution of calls and the age of these calls over the set of services that compose the SBP. A TdPN system models a particular distribution of calls over the services of a deployed SBP. A marked TdPN is a net system $S = \langle N, M_0 \rangle$ where $N$ is a TdPN and $M_0$ is an initial marking on $N$ where all tokens have the age 0.

**Definition** Given a net system $S = \langle N, M \rangle$ and a transition $t$, we say that $t$ is fireable in the marking $M$ by tokens $v = \{(p, x_p) | p \in^\bullet t \wedge x_p \in M(p) \wedge x_p \in I_p\}$, noted by $M[t\rangle^v$, if:

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

As a consequence to this restriction:

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

- If the age of the token is lower than the lower limit of the time interval associated to an arc connecting its place to a transition, the token cannot be used to fire the transition and must wait for its age to increase.
- If the age of the token is higher than the upper limit of the time interval associated to an arc connecting its place to a transition, the token will never be used to fire the transition. If the token can never be used to fire any transition, it will be considered as an outdated and unusable token.

**Definition** Let $M$ be a marking and $t$ a transition, with $M[t\rangle^v$ for some $v$. The firing of the transition $t$ in $M$, noted by $M[t\rangle^v M'$, changes the marking to $M'$ s.t.

$$M' = \begin{cases} M(p) \setminus \{x_p\} & \forall p \in {}^\bullet t \\ M(p) \cup \{0\} & \forall p \in t^\bullet \\ M(p) & \forall p \notin ({}^\bullet t \cup t^\bullet) \end{cases}$$

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

### 3.5.2 Elasticity Operations

Herein, we extend our elasticity operations (duplication/consolidation) to consider timed SBPs.

#### Place Duplication

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

- $P' = P \cup \{p^c\}$
- $T' = T \cup T''$ 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 $T$).
- $Pre'$ (resp. $Post'$): $P' \times I \times T'$ (resp. $T' \times P'$)
- $\equiv_{P'} \subseteq P' \times P'$ 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.
- $M' : P' \to \mathcal{B}(\mathbb{R}_{>=0})$ 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 follows:

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

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

### Place Consolidation

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

- $N'$: is the net $N$ after removing the place $p^c$ and the transitions $(p^c)^\bullet \cup^\bullet p^c$
- $M' : P' \to \mathcal{B}(\mathbb{R}_{>=0})$ with $M'(p) = M(p) + M(p^c)$ and $M'(p') = M(p')$ if $p' \neq p$.

**Example** Figure 3.6-(a) represents the timed SBP model (empty marking) of the online computer shopping service of Figure 3.1. In this model, temporal constraints are represented by time intervals that represent the values that tokens must have to be consumed by a transition firing. For example, a token in the place $s1_1$ must have an age values between 2 and 3 to be consumed by the transition $t1_1$. Figure 3.6-(b) is the resulting system from the duplication of the service $s2_1$ in (a), $D((a), s2_1, s2_2)$. Figure 3.6-(c) is the consolidation of the service $s2_1$ in its copy $s2_2$, $C((b), s2_2, s2_1)$.

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

- $T(x)$: action for the elapse of $x$ unit of time.
- $x'R(t)$: action that fires $x$ time the transition $t$ .
- $D(s)$: action that duplicates the service $s$.
- $C(s, s')$: action that consolidates the service $s'$ in its copy $s$.

A possible execution of the SBP of Figure 3.6-(a) with an initial marking $M_0 = ((s1_1, 0))$ (one token with the age 0 in the place $s1_1$) is given bellow:

- $(s1_1, 0) \xrightarrow{T(2)} (s1_1, 2)$

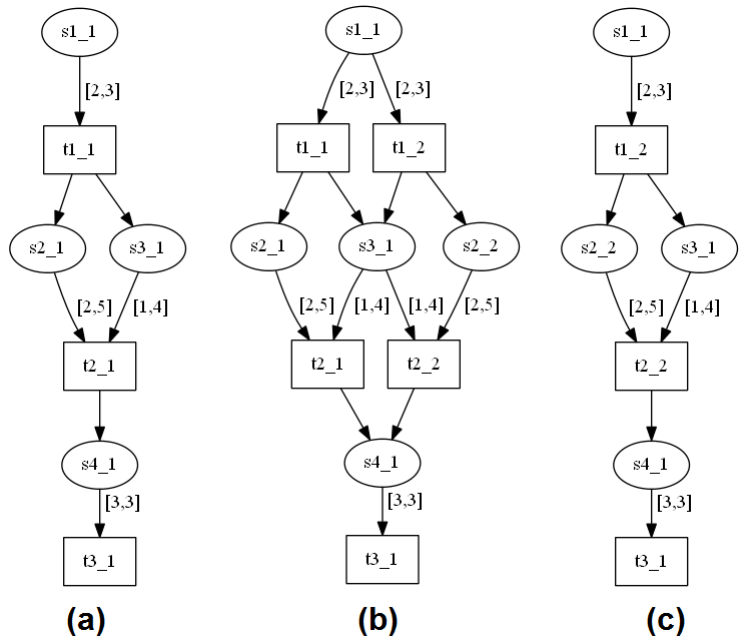


Figure 3.6: An example of the elasticity of the timed SBP of the online computer shopping service.

- $(s1_1, 2) \xrightarrow{1'R(t1_1)} (s2_1, 0), (s3_1, 0)$
- $(s2_1, 0), (s3_1, 0) \xrightarrow{T(2)} (s2_1, 2), (s3_1, 2)$
- $(s2_1, 2), (s3_1, 2) \xrightarrow{1'R(t2_1)} (s3_1, 0)$
- $(s3_1, 0) \xrightarrow{T(3)} (s3_1, 3)$
- $(s3_1, 3) \xrightarrow{1'R(t3_1)} ()$

### Correctness of Elasticity Operations

In the previous section 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 duplication/consolidation properties preserve the semantics of Timed SBPs.

In this section, we presented our formal model for SBPs elasticity. We showed that our model can be used with different types of SBPs (stateless, stateful and timed). In addition, we defined two elasticity operations (duplication/consolidation) that operate

at the scope of services and we proved their correctness in preserving the semantics of SBPs. In the next section, we will propose our approach to intertwine our elasticity operations with an elasticity controller in order to ensure SBPs elasticity.

## 3.6 Generic Controller for SBPs Elasticity

Among others, there are two main approaches for managing 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 [28]. This approach dedicates a controller for each service of the SBP but changes the nature of these latter. The second approach that we adopt in this work consists in setting up a controller that continuously analyzes SBPs execution and possibly generates reconfiguration actions in order to enforce elasticity of deployed SBPs. One can assign a single controller for all the deployed processes, a controller for each tenant (that corresponds to an enterprise) or even a controller for each deployed process. We point out that the choice of the best deployment scenario needs a more thorough study that we consider in our future work.

### 3.6.1 Elasticity Controller

A controller is usually represented by a control loop that consists on harvesting monitoring data, analyzing them and generating reconfiguration actions to correct violations (self-healing and self-protecting) or to target a new state of the system (self-configuring and self-optimizing) [53]. In our work, we are interested in setting up a controller to ensure SBPs elasticity. Therefore, we propose to intertwine the elasticity operations (duplication/consolidation) previously defined with a controller to ensure SBPs elasticity. In this control loop (see Figure 3.7), the central element represents the SBP for which we want to ensure elasticity. For this loop we need to collect information about the SBP execution (workload, time response etc.). Afterward, we need to analyze and reason about these information to decide on triggering elasticity actions. We need also reconfiguration actions (duplication/consolidation) to carry out changes to the SBP in order to ensure its elasticity. The different functions of the elasticity control loop are defined as:

- Monitor function that collects, aggregates, filters and reports monitoring data collected from the SBP;
- Analyze function that provides the mechanisms that correlate and model complex situations and allow the controller to interpret the environment and the state of the system according to some elasticity strategies;
- Reconfiguration function that provides the mechanisms that control the execution of reconfiguration actions (duplication/consolidation) needed to ensure the SBP elasticity.

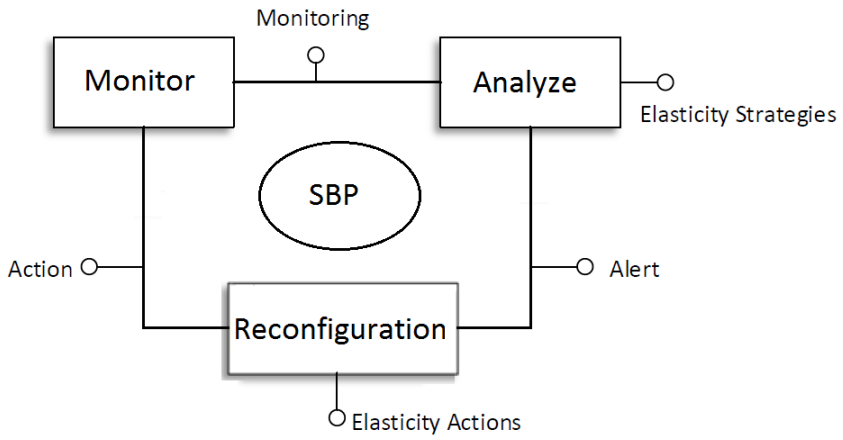


Figure 3.7: An overview of the Control Loop for SBPs elasticity.

Several strategies can be used [54, 21, 55] to manage SBPs elasticity. A strategy is responsible of making decisions on the execution of elasticity mechanisms *i.e.,* deciding when, where and how to use these mechanisms. Generally, these strategies can be predictive or reactive. Reactive strategies are based on Event-Condition-Action rules while predictive strategies are based on predictive-performance models and load forecasts. For example, if the monitoring data is related to a service response time, we can specify a strategy that raises an alert if the value of the response time is over a given threshold. Examples of these aspects will be detailed later in the next chapter.

The abundance of possible strategies requires a generic solution which allows the implementation and execution of different elasticity strategies. For this reason, we propose a framework, called generic Controller, for SBPs elasticity in which different elasticity strategies can be implemented. Our generic 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 if and only if 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 increase. The duplication action can be executed as many times as necessary to get a sufficient number of service copies in order to ensure the QoS of the SBP by avoiding resources under-provisioning.

- Consolidation: Is about the deletion of an unnecessary copy of a service in order to meet its workload decrease. The consolidation action can be executed

as many times as necessary to get the minimal and optimal number of service copies in order to avoid resources over-provisioning while maintaining the QoS of the SBP.

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.

In order to manage timed SBPs, we add to the controller a fourth action called Time elapse. This action is about the time advancement on all tokens (calls) over the services of the considered SBP. Adding this transition in our generic controller allows the use of SBP model with or without time constraints (place/transition, colored and timed Petri net model). Note that this action is used only with Timed SBPs.

### 3.6.2 Formal Description of the Generic Controller

As pointed out in the previous section our goal here is not to propose an additional elasticity strategy, but a framework, called generic controller, to implement and execute different strategies. We propose to model the controller as a high level Petri net (HLPN). As classical 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, if and only if 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 3.8. The controller contains one place (BP) of type net system. The marking of this place evolves either by calls transfer (Routing), by the elapse of time (Time elapse), by the duplication of an overloaded service (Duplication) or by the consolidation of an underused service (Consolidation). These four transitions (actions) are guarded by conditions that decide when, where and how to perform these actions:

- 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

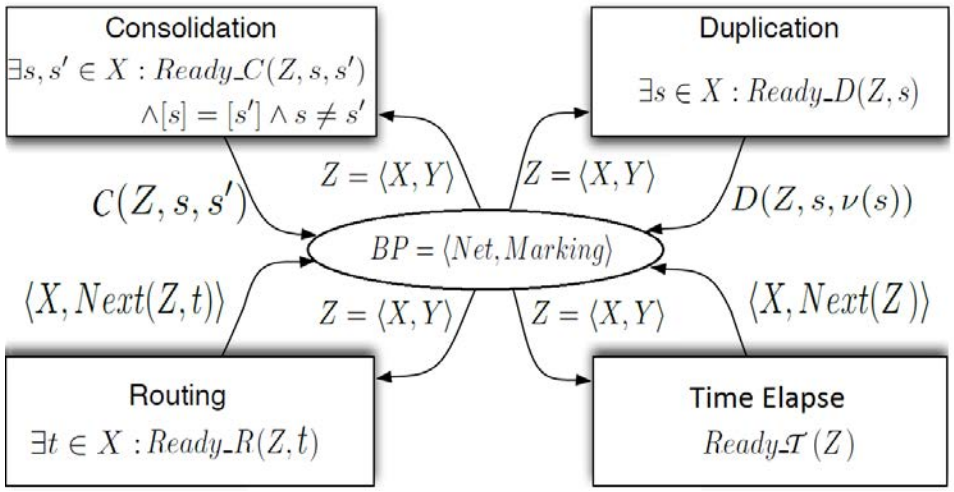


Figure 3.8: HLPN model of the generic controller

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 $ready_D(Z, s)$ is satisfied. The firing of the Duplication transition adds a new net 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 $ready_C(Z, s, s')$ is satisfied. The firing of the Consolidation transition adds a net system resulted from the consolidation of $s'$ in $S$.

- Time elapse: This transition is fireable if we can bind the variable $Z$ to a net system $S = \langle N, M \rangle$ where the predicate $ready_T(Z)$ is satisfied. The firing of the Time elapse transition adds a net system resulted from the elapse of a time unit in $S$.

The execution of these actions is performed after checking the guards of the execution of these actions ($ready_R$, $ready_D$, $ready_C$, $ready_T$). 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 condition that decides how to increment time is implemented in the predicate $ready_T$. In our controller, the conditions ($Ready_D$, $Ready_C$, $Ready_R$) are generic

to allow the use of different elasticity strategies while the condition $Ready_T$ is implemented to increment time. By instantiating the generic controller, one can implement and execute different elasticity strategies.

### 3.6.3 Illustrating Example of SBPs elasticity

In order to illustrate our formal model for SBPs elasticity, we propose hereafter to discuss an example scenario of the instantiation of our generic controller. To this end, we propose to instantiate our elasticity controller to manage the elasticity of the SBP example previously defined (Figure 3.3).

The SBP example is composed by four services (S1, S2, S3 and S4). In our example, we assume that each service of the SBP is provided by a maximum threshold capacities (noted $Max_t$) that indicate when a service is over-loaded. Note here that these thresholds represent the maximum number of calls on each service.

- Max_t($s1$) = 8. Max_t($s2$) = 5. Max_t($s3$) = 10. Max_t($s4$) = 7.

In order to manage the SBP elasticity, we propose in this example to use an elasticity strategy that uses the load of services as a scaling indicator to take its elasticity decisions. To do this, we have to instantiate the three generic predicate $Ready_D$, $Ready_C$, $Ready_R$:

- $Ready_D(S, s) : M(s) \geqslant Max_t(s) \wedge \nexists s' \in [s] : M(s') < Max_t(s')$.
  It duplicates a copy $s$ of service if all copies of this service have already reached their maximal threshold.

- $Ready_C(S, s', s) : M(s') = 0 \wedge M(s) < Max_t(s)$.
  It consolidates a copy $s'$ of service if this copy does not contain calls and there is another copy $s$ of the service that has not reached its maximum threshold.

- $Ready_R(S, t) : \forall s \in P : M'(s) < Max_t(s)$ with $M[t\rangle M'$.
  It routes a call if this call transfer does not cause a violation of the maximum thresholds of services.

When the controller is instantiated (i.e., time $t0$), the SBP has an empty marking ($M(S1)$, $M(S2)$, $M(S3)$, $M(S4)$)=(0, 0, 0, 0) which means that there is no invocation of the SBP (see Figure 3.9). The controller observes the evolution of the SBP marking and checks whenever an elasticity condition ($Ready_D$ or $Ready_C$) is verified in order to execute an elasticity action (duplication/consolidation).

Assuming that at a certain time $t1$, the SBP is invoked. The invocation consists in adding tokens in the place $S1$ (each token represents a service call). We suppose that the marking of the SBP evolves and leads to the following marking ($M(S1)$, $M(S2)$, $M(S3)$, $M(S4)$)=(2, 4, 4, 0) which means that there are calls being proceeded on some services of the SBP (As shown in Figure 3.10, 2 calls on the service $S1$ and 4

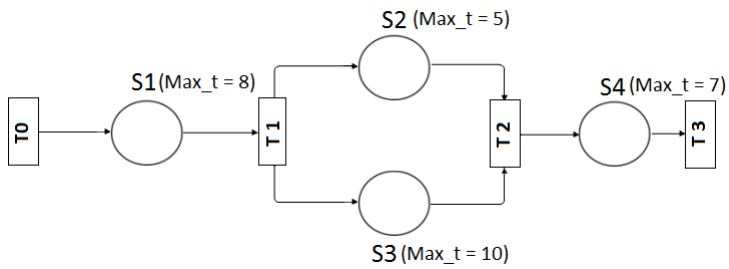


Figure 3.9: The initial SBP managed by the elasticity Controller

calls on services $S2$ and $S3$). At this time, the controller analyzes this marking in order to check if there is an elasticity action to perform. Since there is no condition verified, no elasticity action is performed.

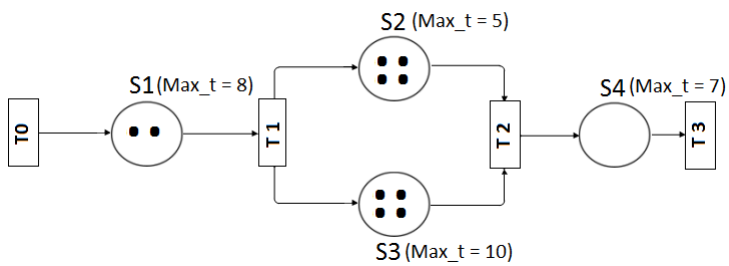


Figure 3.10: The SBP at time $t1$.

Assuming that at a certain time $t2$, the marking of the SBP is ($M(S1)$, $M(S2)$, $M(S3)$, $M(S4)$) = (2, 5, 5, 2) as shown in Figure 3.11-(a). At this time, the duplication condition *Ready_D* is verified (i.e., the load of the service $s2$ has reached its maximal threshold). Consequently, the controller performs a duplication action that adds a new copy of the service $S2$ (see Figure 3.11-(b)). As we can see, the new marking of the SBP is ($M(S1)$, $M(S2)$, $M(S2')$, $M(S3)$, $M(S4)$)=(2, 5, 0, 5, 2). Note here that the marking of the new copy is empty.

Assuming that at a certain time $t3$, the marking of the SBP is ($M(S1)$, $M(S2)$, $M(S2')$, $M(S3)$, $M(S4)$)=(0, 5, 2, 7, 2) as shown in Figure 3.12. At this time, the controller analyzes this marking and does not perform an elasticity action (no condition is verified).

Assuming that at a certain time $t4$, the marking of the SBP evolves and reaches the marking $(M(S1), M(S2), M(S2'), M(S3), M(S4))$=$(0, 3, 0, 3, 4)$ as shown in Figure 3.13-(a). At this time, the consolidation condition *ready_C* is verified (i.e., there is an empty copy of the service $S2$ and another copy of this service that has not reached

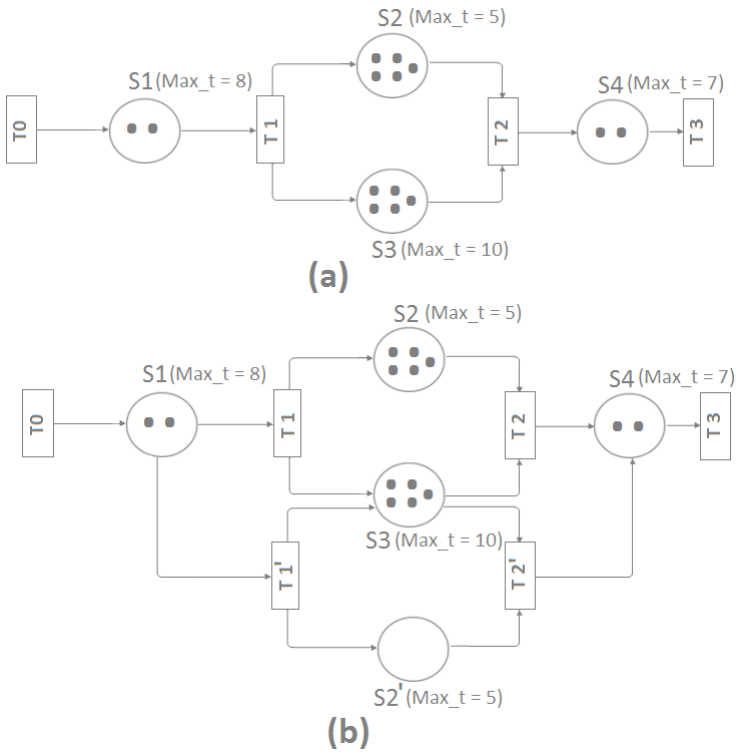


Figure 3.11: Duplication of the service S2 at time $t2$.

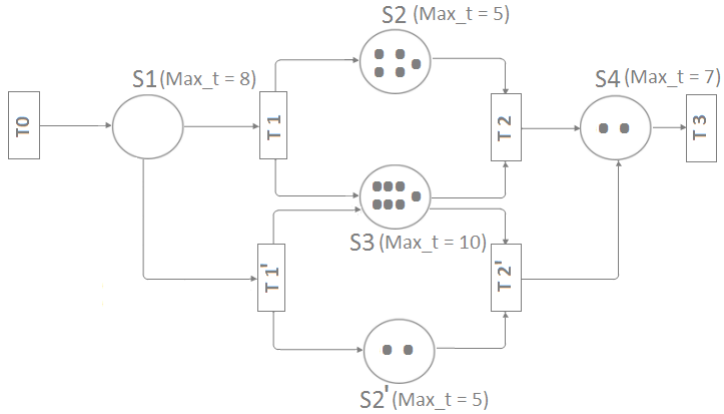


Figure 3.12: The SBP at time $t3$.

its maximal threshold). Consequently, the controller performs a consolidation action that removes the empty copy of the service $S2$ (see Figure 3.13-(b)) which produces

the following marking ($M(S1)$, $M(S2)$, $M(S3)$, $M(S4)$)=(0, 3, 3, 4).

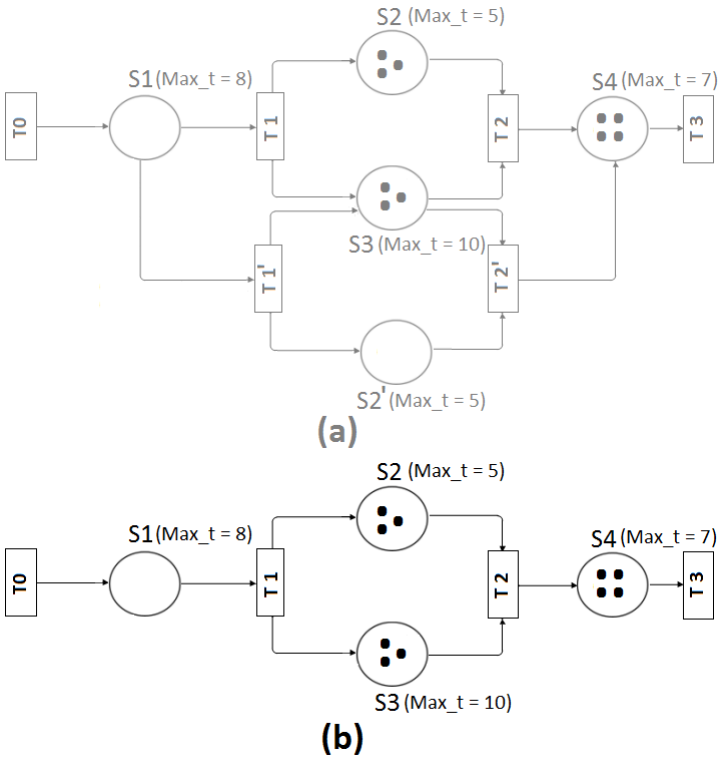


Figure 3.13: Consolidation of the service S2 at time $t4$.

The execution of the instantiated controller continues, according to the implemented elasticity strategy, until there are no calls to process.

## 3.7 Conclusion

In this chapter, we proposed a formal model for SBPs elasticity that intertwines duplication/consolidation operations with an elasticity controller. We modeled SBPs using Petri nets and formalized elasticity operations (duplication/consolidation) that operate at the scope of services. We showed that our formal model ensures the elasticity of stateless, stateful and timed SBPs while preserving the semantics of SBPs. In addition, we proposed to intertwine our elasticity operations with a generic controller that monitors SBP execution, analyzes monitoring information and executes elasticity actions (duplication/consolidation) in order to ensure SBPs elasticity.

After facing the challenge of defining a model for ensuring SBPs elasticity at the scope of services, we were interested in the evaluation of elasticity strategies. Elasticity

strategies are used to decide on when, where and how to use duplication/consolidation operations. In fact, several strategies can be used to manage SBPs elasticity. The abundance of possible strategies requires their evaluation and validation. In the next chapter, we propose to use our generic controller to evaluate elasticity strategies in order to guarantee their effectiveness before using them in real Cloud environments. To this end, we will propose two types of evaluations: verification-based and simulation-based evaluations.

Chapter 4

# Evaluation of Elasticity Strategies

**Contents**


**4.1 Introduction** . . . . . **77**
**4.2 Generic Controller for the Evaluation of Elasticity Strategies** . **78**
**4.3 Validating Elasticity Strategies** . . . . . **80**
4.3.1 Verification-based Evaluation . . . . . 80
4.3.2 Proof of Concept: Validating Elasticity Strategies . . . . . 81
4.3.2.1 Implementation of the generic Controller . . . . . 82
4.3.2.2 Use case . . . . . 84
4.3.2.3 Evaluation of Strategies . . . . . 86
**4.4 Comparing Elasticity Strategies** . . . . . **87**
4.4.1 Performance-based Evaluation . . . . . 88
4.4.2 Reliability-based Evaluation . . . . . 88
4.4.3 Proof of Concept: Comparing Elasticity Strategies . . . . . 89
4.4.3.1 Performance-based Evaluation . . . . . 89
4.4.3.2 Reliability-based Evaluation . . . . . 92
**4.5 Conclusion** . . . . . **98**


## 4.1 Introduction

In the previous chapter, we presented our formal model for ensuring SBPs elasticity. The proposed model intertwines two elasticity operations (duplication and consolidation) with a generic controller for SBPs elasticity. The generic controller monitors SBPs execution, analyzes monitoring information and executes elasticity actions (duplication/consolidation) according to an elasticity strategy in order to ensure QoS of SBPs and avoid over-provisioning and under-provisioning of resources.

Elasticity strategies are responsible of making decisions on the execution of elasticity mechanisms i.e., deciding when, where and how to use duplication/consolidation

operations. These strategies are responsible of guaranteeing provisioning of necessary and sufficient resources that ensure a smooth functioning of the deployed SBP despite of changes in its solicitations. The abundance of possible strategies requires their validation and evaluation to guarantee their effectiveness before using them in real Cloud environments.

In this chapter, we propose to use the previously defined elasticity controller as a framework for the evaluation of different elasticity strategies. For this sake, we start by introducing our approach for using the generic controller for the evaluation of elasticity strategies. Then, we present an approach for validating elasticity strategies using a verification-based approach. This approach allows formal verification of the correctness of elasticity strategies (i.e., the ability of a strategy to satisfy some properties or not). Afterwards, we present an approach for comparing elasticity strategies using a simulation-based approach. This approach allows comparing between different elasticity strategies in terms of performance and reliability in order to choose the most suitable one. For the Proof of Concept, we detail an example of validation and comparison of elasticity strategies using the generic controller. To this end, we present the implementation of the controller with the SNAKES toolkit and we detail the evaluating scenarios that we used as well as the evaluation results.

## 4.2 Generic Controller for the Evaluation of Elasticity Strategies

The generic controller has been designed to be able to manage SBPs elasticity with different elasticity strategies. In our work, we propose to use the controller as a framework for the evaluation of elasticity strategies. Using our generic controller, we can validate and evaluate the effectiveness of these strategies before using them in real Cloud environments. To do so, we have to instantiate and execute the controller with the elasticity strategy we want to evaluate. The execution of the controller allows observing and analyzing the behavior of this strategy.

In order to instantiate the controller, we have to define:

- The SBP to be managed by the controller. A SBP is defined by its structure (services, calls transfer, etc.) and constraints (thresholds, time constraints, etc.).

- The calls arrival on the SBP. A calls arrival represents the clients' invocations of the SBP. It is possible to define the calls arrival by providing an initial marking to the SBP net, 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 or by generating the calls arrival using an arrival law (e.g., random, Poisson process, etc.).

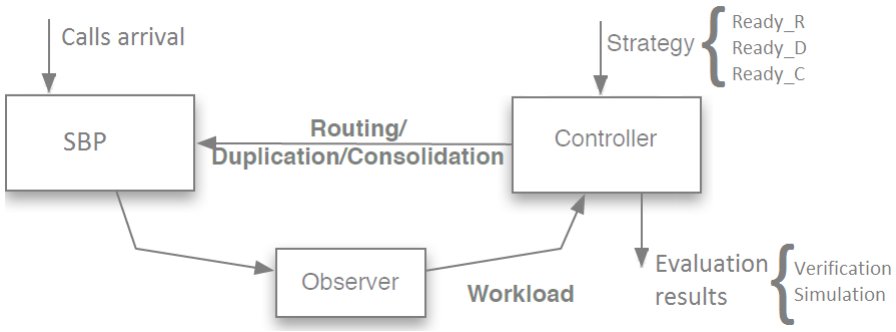


Figure 4.1: Generic Controller for the Evaluation of Elasticity Strategies

- The elasticity strategy to evaluate. An elasticity strategy is defined by instantiating the conditions that trigger each controller action i.e., *Ready_R* for routing, *Ready_D* for duplication and *Ready_C* for consolidation.

Once the controller is instantiated, it will monitor the execution of the SBP and execute the needed actions (Routing, Duplication or Consolidation) in order to ensure the SBP elasticity (see Figure 4.1). At each step of the SBP execution, the controller stores the needed information for evaluating the used elasticity strategy. The execution of the instantiated controller allows analyzing the SBP execution with respect to the implemented strategy, calls arrival and SBP constraints (thresholds, time constraints, etc.). As we will see, analyzing the SBP execution allows checking and observing many properties and indicators related to the SBP elasticity.

In our work, we propose a two-level evaluation approach for the evaluation of elasticity strategies:

- **Validating Elasticity Strategies:** The first level consists in providing a verification-based approach that validates the correctness of an elasticity strategy. By correctness we mean the ability of a given elasticity strategy to verify or not a set of properties (e.g., violation of QoS).

- **Comparing Elasticity Strategies:** The second level consists in providing a simulation-based approach for comparing different elasticity strategies. By comparing strategies we mean the ability to indicate the most efficient strategy (e.g., in terms of performance).

It is important to note here that the evaluation results are related to the SBP used in the evaluation process while considering the SBP characteristics and the calls arrival scenario. Changing one of these parameters can produce different evaluation results.

Considering this, our approach allows a SBP owner to evaluate different elasticity strategies in order to choose the most adequate one according to its SBP. On one hand,

the verification-based approach allows performing a qualitative evaluation of elasticity strategies by using formal methods to verify if these strategies ensure some properties or not (i.e., validation). On the other hand, the simulation-based approach allows performing a quantitative evaluation of elasticity strategies by calculating indicators related to SBPs elasticity (e.g., performance indicators). Coupling verification and simulation allows having a global view about strategies effectiveness in ensuring SBPs elasticity.

## 4.3 Validating Elasticity Strategies

In this section we aim at validating elasticity strategies. Validating a strategy consists in formally verifying if this strategy satisfies some properties or not. To do this, we propose a verification-based approach to validate elasticity strategies. In addition, we present a proof of concept of the validation of elasticity strategies.

### 4.3.1 Verification-based Evaluation

In order to validate elasticity strategies, a verification-based approach can be used to perform qualitative evaluations of elasticity strategies. In this approach, we can generate using a HLPN tool the reachability graph of the elasticity controller in order to analyze it with a model-checker and verify certain properties. 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 CPN nets but do not support any property [56].

The reachability graph [57] is the set of all system states reachable from the initial state of a given Petri net. It can be represented as a directed graph that represents all reachable states and state changes of the system. The reachability graph can be used to answer a large set of analysis and verification questions concerning the behavior of the system such as absence of deadlocks, the possibility of always being able to reach a given state, and the guaranteed delivery of a given service. One of the most popular analysis approaches is to specify and check requirements by inspecting the reachability graph, e.g., model checking approaches [58].

Model checking [59] is an automated technique for the verification of systems that are modeled as finite state models. It examines all possible system scenarios in a systematic manner in order to verify that a given system model satisfies a certain property or not. Model checking is used to examine all system states to check whether these states satisfy a given property. If a state is encountered that violates the considered property, the model checker provides a counterexample that indicates how the model could reach the undesired state. The counterexample describes an execution path that leads from the initial system state to a state that violates the property being verified.

In our work, the controller reachability graph contains all the possible evolutions of the SBP with respect to SBP constraints, calls arrival and the implemented elasticity strategy. The resulted graph can be analyzed using a model-checking approach to perform verification-based evaluations of elasticity strategies. A simplified example of the reachability graph that can be generated for a given elasticity strategy is shown in Figure 4.2. Several properties can be verified using model checking, these properties can be generic properties (e.g., deadlock) or properties related to elasticity (e.g., elasticity loop). In our work, we propose to verify some significant examples of properties that are related to SBPs elasticity. These 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. This strategy modifies the semantics of SBPs (i.e., transition firing), it would be interesting to 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 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 which means that the strategy causes unnecessary duplication of services.

We point out that other properties (generic or related to elasticity) can be studied and verified in the verification-based approach. In our work we aim at providing a model that generates the reachability graph of the controller according to an elasticity strategy. This model can then be used by existing verification tools in order to perform verify other properties.

The verification-based approach allows performing qualitative evaluations of elasticity strategies using model-checking techniques. This evaluation approach allows verifying properties related to SBPs execution and/or elasticity.

### 4.3.2 Proof of Concept: Validating Elasticity Strategies

We present hereafter an example, for the proof of concept, of elasticity strategies validation with the controller. To this end, we start by introducing the implementation of the generic controller with the *SNAKES* toolkit. Then, we present the use case

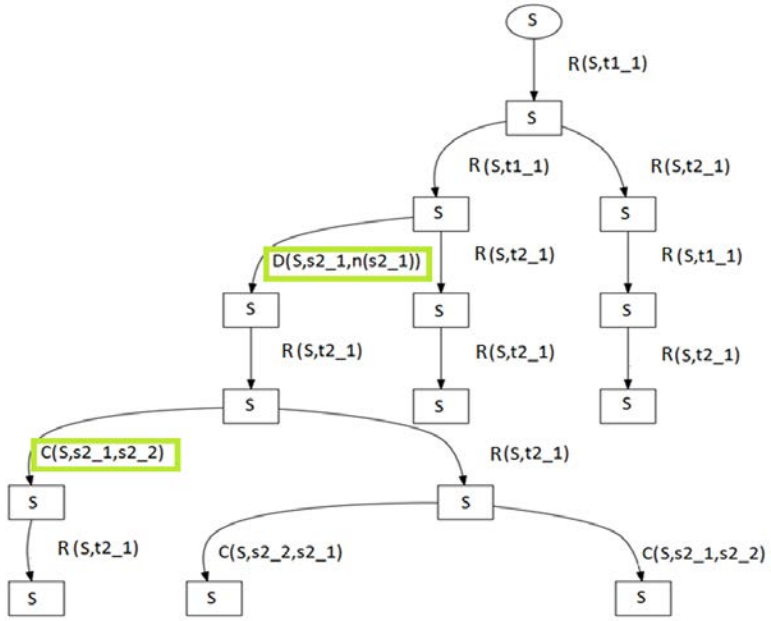


Figure 4.2: An example of the SNAKES reachability graph of an elasticity strategy

that we used for our evaluations. Afterwards, we detail and analyze the evaluation results of these strategies using the verification-based approach.

#### 4.3.2.1 Implementation of the generic Controller

In order to implement the controller, we propose to use the *SNAKES* toolkit [60]. The *SNAKES* toolkit is a general Petri net library designed with quick prototyping in mind. To do this, *SNAKES* offers a flexible architecture based on a core library, that defines a basic Petri net structure, complemented with a variety of extension modules (i.e., plugins), that introduce additional features. The core library provides a general model of Python-colored Petri nets: tokens are Python objects, transitions guards are Python expressions and arcs can carry Python expressions.

The *SNAKES* toolkit allows defining a Petri net whose tokens are Petri nets (nets in nets). We propose to implement our elasticity controller as a Petri net that will have the SBP net as token. The transition of the controller (Routing, Duplication and Consolidation) can synchronize the SBP transition firing (Routing) and modify the structure of the SBP net (place duplication/consolidation).

Hereafter we present how we can implement the elasticity controller using *SNAKES*. The first step consists in defining the SBP net to control. For this, we have to define the SBP net places, transitions and arcs.

```
def token_net (name) :

    # SBP places
    net.add_place(Place("s1_1", []))
    net.add_place(Place("s2_1", []))
    ...

    # SBP transitions
    net.add_transition(Transition("t1_1"))
    net.add_transition(Transition("t2_1"))
    ...

    # SBP arcs
    net.add_input("s1_1", "t1_1", Value(dot))
    net.add_output("s2_1", "t1_1", Value(dot))
    ...

    return net
```

After defining the SBP net, we define the controller net that will manage the SBP net. The controller net is composed by a single place (BP) that will contain a token that represents the SBP net. The controller net transitions represent the three controller actions (routing, duplication and consolidation).

```
n = PetriNet("controleur")

# The SBP
n.add_place(Place("BP", [token_net("SBP")]))

# Routing
n.add_transition(Transition("routing",
                                Expression("ready_R(S,t)")))
n.add_input("BP", "routing", Variable("S"))
n.add_output("BP", "routing", Expression("R(S,t)"))

# Duplication
n.add_transition(Transition("duplication",
                                Expression("ready_D(S,s)")))
n.add_input("BP", "duplication", Variable("S"))
n.add_output("BP", "duplication",
                        Expression("D(S,s,new_copy(S,s))"))
```

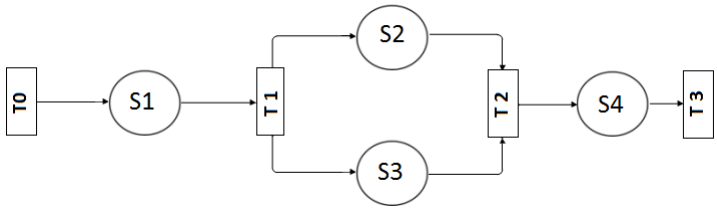


Figure 4.3: Petri net of the SBP of the online computer shopping service.

```
# Consolidation
n.add_transition(Transition("consolidation",
                            Expression("ready_C(S,s,sc)")))
n.add_input("BP", "consolidation", Variable("S"))
n.add_output("BP", "consolidation", Expression("C(S,s,sc)"))
```

As we can see, each transition is guarded by a firing condition that represents the condition of executing this action ($ready_R$ for the routing strategy, $ready_D$ for the duplication strategy and $ready_C$ for the consolidation strategy). If one of these conditions is verified, the transition is fired and the SBP net will be modified. For example, in the case of duplication, if the duplication strategy ($ready_D$) is verified ($ready_D(S, s)$ return true), the duplication transition will consume the SBP net and generate a new SBP net which is the resulted net after the duplication of the service $s$ ($D(S, s, new_copy(S, s))$).

#### 4.3.2.2 Use case

To instantiate the generic controller, we have to define the SBP to be controlled along with the elasticity strategy and the calls arrival scenario that specifies the way in which calls arrive to the SPB. Then, we have to implement the elasticity strategies to evaluate.

We propose in this experiment to apply our elasticity approach on the SBP example of the online computer shopping service (Figure 4.3) presented in chapter 3. This SBP example is composed by 4 services ($s1, s2, s3$ and $s4$) with $M_0 = (0, 0, 0, 0)$ as its initial marking. We assume in this example that each service of the SBP is provided by a maximum and minimum threshold capacities ($Max_t$ and $Min_t$). Above the maximum threshold the QoS would no longer be guaranteed and under the minimum threshold we have an over allocation of resources. Note here that these thresholds represent the maximum number of running instances (calls) on each service. For this experiment, we used the following thresholds:

- Max_t($s1$) = 5. Max_t($s2$) = 3. Max_t($s3$) = 5. Max_t($s4$) = 5.
- Min_t($s1$) = 1. Min_t($s2$) = 1. Min_t($s3$) = 1. Min_t($s4$) = 1.

An invocation (a call) of the SBP is represented by adding a token to a copy of the place $s1$, the invocation takes end by removing a token from a copy of the place $s4$. As explained previously, the calls arrival can be defined by providing an initial marking to the SBP net, by adding a new transition in the controller that simulates the arrival of calls or by generating the calls arrival using an arrival law. In this use case, we define the calls arrival on the SBP using a Poisson process with mean 2.

To decide if an elasticity action is needed or not, an elasticity strategy needs to base its decision on some metrics (scaling indicators). In this example, we propose to use load of services as metric (scaling indicator) to base our elasticity strategies. The load of a service is represented by the number of tokens (marking) of its corresponding place in the SBP net. Note here that it is also possible to use the response time as a scaling indicator. To do this, we have to define a function that estimate the service response time according to its load (the marking of the considered service).

In this use case, we propose to evaluate two strategies (Strategy 1 and Strategy 2) inspired from the literature [61, 62]. In addition, we propose to evaluate a third strategy (Strategy 3) which is defined by us in order to illustrate the interest of our evaluation approaches. The definition of a strategy consists in instantiating the three generic predicates $ready_R$, $ready_D$ and $ready_C$ (the predicate $ready_T$ is already instantiated to increment the age of SBP tokens in the case of timed SBPs). Hereafter the definition of the three strategies:

**Strategy 1**

In [61] 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 service $s$ if all copies of this service have already reached their maximal threshold. In addition, there is a call waiting to be transferred to this service $s$.

- $Ready_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min_t(s) \wedge \nexists t \in^{\bullet} [s] : M[t\rangle$.
  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 no 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 [62] 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 their 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 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 $M[t\rangle M'$.
  It uses the same routing strategy than strategy 1.

**Strategy 3**

We define a third strategy that implements only a routing strategy.

- $Ready_D(S, s) : false$.
  No duplication.

- $Ready_C(S, s', s) : false$.
  No consolidation.

- $Ready_R(S, t) : \forall s \in P : M'(s) < Max_t(s)$ with $M[t\rangle M'$.
  It uses the same routing strategy than strategies 1 and 2.

#### 4.3.2.3 Evaluation of Strategies

Now that we have defined the use case, we will use our generic controller to validate the defined elasticity strategies. To do this, we will evaluate these strategies using a verification-based approach that consists in generating the reachability graph of the instantiated controller and analyzing this graph using model-checking techniques.

Once the controller is instantiated, we generate the reachability graph that contains all the possible evolutions of the SBP with respect to SBP constraints, calls arrival and used strategy. The analysis of this reachability graph generated allows deducing some behavioral properties of the execution of the SBP controlled. These properties are summarized in the Table 4.1.

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.

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

Table 4.1: Verification-based evaluation of elasticity 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 these 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.

As we can see in this use case, the verification-based approach allows the validation of the strategy 1 according to the three properties that we have defined (QoS violation, Blocked services and Elasticity loop).

## 4.4 Comparing Elasticity Strategies

Once the elasticity strategies are formally validated in terms of correctness, it is interesting to be able to compare between these strategies to choose the most suitable one considering the SBP to manage. In our work, we propose an approach to compare elasticity strategies in terms of performance (resource consumption/response time) and reliability (loss of calls/response time). To this end, we propose a simulation-based approach for quantitative evaluations of elasticity strategies.

Simulation is a technique for analyzing dynamic systems. It allows understanding the dynamic behavior of the simulated system, comparing scenarios, evaluating different control strategies and optimizing performance. We propose in our work to use simulation to study and compare the behavior of different elasticity strategies. For this, we will simulate the SBP execution according to some elasticity strategies in order to evaluate them.

The simulation-based approach allows evaluating elasticity strategies by defining a set of indicators to analyze strategies' behavior. 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. The analysis of these

indicators allows evaluating strategies' behavior. Many parameters can be observed and evaluated, we propose in our work two kind of evaluation based on simulation: performance-based and reliability-based evaluations.

### 4.4.1 Performance-based Evaluation

The first kind of simulation-based evaluation that we propose consists in a performance-based evaluation. In this kind of evaluation, we study the SBP performance with different elasticity strategies. To perform this evaluation, we focus on two parameters in order to answer two questions:

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

The average response time of the SBP can be calculated using the average 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.

### 4.4.2 Reliability-based Evaluation

The second kind of simulation-based evaluation consists in a reliability-based evaluation. In this kind of evaluation, we study the influence of elasticity strategies on the reliability of SBPs in terms of the ability of the strategy in treating SBP calls with a minimal loss of calls. To perform this evaluation, we define two indicators:

- Call processing time: The time required for processing the received SBP calls. We calculate the time required for the processing of each received call. This indicator provides information about the time necessary for SBP calls processing. Using this indicator we can observe the influence of the implemented strategy on the SBP processing time according to the evolution of calls arrival.
- Loss of calls: The loss of calls is due to the outdated calls that can be caused by the temporal constraints on the SBP during its execution (expired calls that will never be treated). We calculate for each execution, the number of lost calls according to calls arrival. This indicator provides information about the

proportion of lost calls. Using this indicator we can observe the influence of the implemented strategy on the proportion of lost calls according to the evolution of calls arrival.

The simulation-based approach allows performing quantitative evaluations of elasticity strategies by calculating performance indicators (resource consumption, processing time, etc.) related to SBPs execution/elasticity.

### 4.4.3 Proof of Concept: Comparing Elasticity Strategies

Hereafter we propose to compare elasticity strategies using simulation. To this end, we evaluated these strategies using two approaches: performance-based and reliability-based evaluations.

In this proof of concept, we use the implementation of the generic controller with the SNAKES toolkit described in Section 4.3.2.

#### 4.4.3.1 Performance-based Evaluation

In this experiment, we used the SBP system of Figure 4.3 with $M_0 = (0, 0, 0, 0)$ as its initial marking. The calls arrival on the SBP is defined using a Poisson process with mean 2. This scenario was applied on the strategies 1 and 2 defined in Section 4.3.2. For this evaluation we used the following thresholds:

- $\text{Max_t}(s1) = 5$. $\text{Max_t}(s2) = 3$. $\text{Max_t}(s3) = 5$. $\text{Max_t}(s4) = 5$.
- $\text{Min_t}(s1) = 1$. $\text{Min_t}(s2) = 1$. $\text{Min_t}(s3) = 1$. $\text{Min_t}(s4) = 1$.

Once the controller is instantiated and the performance indicators are defined, the values of these indicators will be calculated according to the evolution of the controller. In the performance-based approach, these indicators represent the resource consumption and the average response time according SBP constraints, calls arrival and used strategy (strategy 1 or strategy 2). The analysis of these performance indicators allows the evaluation of strategies' performance in terms of resource consumption and average response time.

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 Figures 4.4 and 4.5. 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 (i.e., the condition of routing). This guarantees the availability of resources to provide required QoS and avoid resources over-utilization.

Figure 4.6 represents the evolution of average response time on one possible execution of the controller. We notice a difference between the strategies in the average response time. We can see that strategy 2 ensures a better average response time than

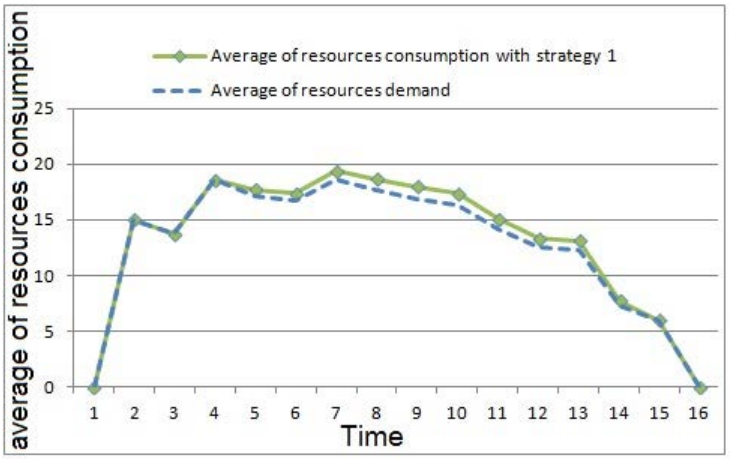


Figure 4.4: The average evolution of resources consumption with strategy 1

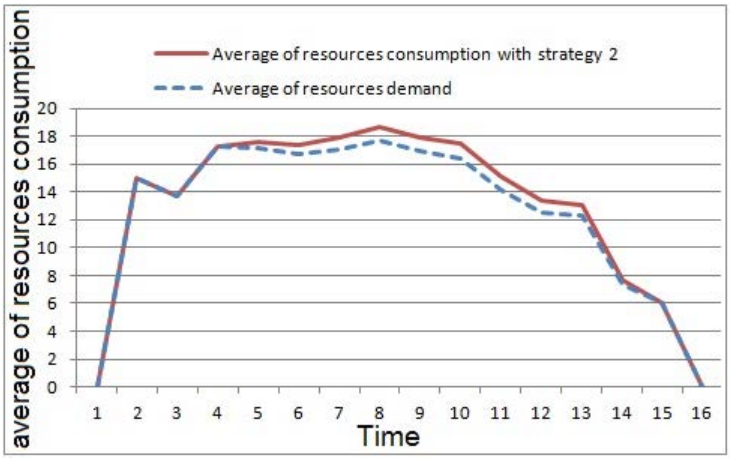


Figure 4.5: The average evolution of resources consumption with strategy 2

the strategy 1. In fact, The strategy 2 is more reactive than the strategy 1 which causes more duplication/consolidation than strategy 1. The reactivity of the strategy 2 ensures a better average response time compared to strategy 1. The evolution of resources consumption on one possible execution of the controller is shown in Figure 4.7. 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. The strategy 2 ensures a better average response time but consume more resources than

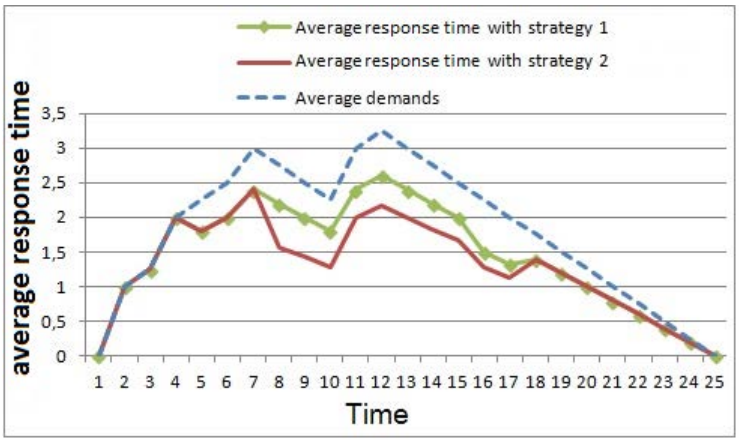


Figure 4.6: The evolution of average response time according to the average of demands

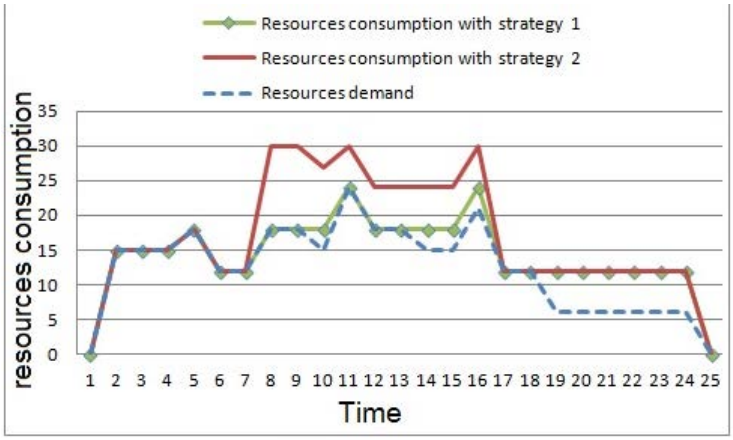


Figure 4.7: The evolution of resources consumption according to the demand of resources

strategy 1. 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 (i.e., the condition on the existence of service call waiting to be transferred). So, the controller using strategy 2 reacts faster. Note here that the reactivity of strategy 2 does not always mean better efficiency. In fact, this reactivity can cause unnecessary duplication of services.

#### 4.4.3.2 Reliability-based Evaluation

In this experiment, we used the SBP system with time constraints shown in Figure 4.8 and we vary the initial marking of the SBP in order to simulate the calls arrival on the SBP. This scenario was applied on elasticity strategies in order to observe the evolution of the processing time and the proportion of lost calls during the SBP execution. In this evaluation we used the following thresholds:

- Max_t($s1$) = 20. Max_t($s2$) = 3. Max_t($s3$) = 15. Max_t($s4$) = 15.
- Min_t($s1$) = 1. Min_t($s2$) = 1. Min_t($s3$) = 1. Min_t($s4$) = 1.

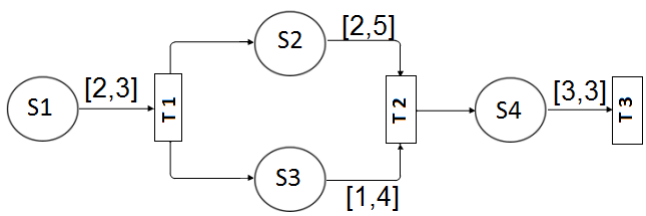


Figure 4.8: Petri net of the Timed SBP of the online computer shopping service.

Duplication/consolidation can be time-consuming operations. Therefore, it is necessary to consider the cost of these operations in the evaluation results. We point out that defining the cost of duplication and consolidation operations needs a separated treatment. In this example, we assume that the duplication has a temporal cost defined as $T_dup = 1$ (time required to create a new copy of a service). On the other hand, we assume that the consolidation is not a consuming time operation (no consumption of time for the removal of an empty copy of a service), consequently we set the temporal cost for the consolidation as $T_con = 0$.

In order to illustrate the reliability-based evaluation approach, we define a fourth strategy that implements an algorithm which takes its elasticity decision depending on the load of services and the way the load can be transferred between services.

**Strategy 4**

- $Ready_D(S, s) : \exists s' \in^{\bullet} ({}^{\bullet}s) : M([s']) > Max_t([s]) - M([s])$.
  It duplicates a copy $s$ of service if there is a service $s' \in^{\bullet} ({}^{\bullet}s)$ where the load of all copies of this service can overload the copies of service $s$ when this load will be transferred to these copies.
- $Ready_C(S, s', s) : M(s') = 0 \wedge M(s) \leqslant Min_t(s) \wedge \nexists s'' \in^{\bullet} ({}^{\bullet}s') : M([s'']) > Max_t([s]) - M([s])$.
  It consolidates a copy $s'$ of service if this copy does not contain calls and there is another copy $s$ that has not reached its minimum threshold. In addition, we

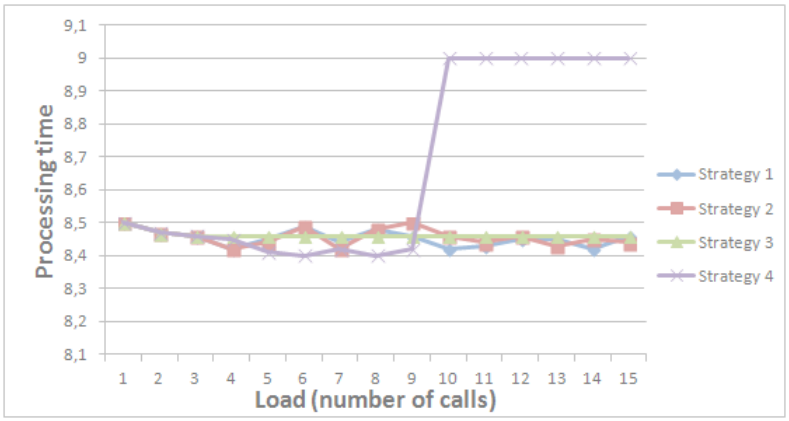


Figure 4.9: the evolution of the minimal calls processing time according to load variation

check that there is not a service $s'' \in^{\bullet} ({}^{\bullet}s)$ where the load of all its copies can overload the copies of the service $s$ when this load will be transferred to these copies.

- $Ready_R(S,t) : \forall s \in P : M'(s) < Max_t(s)$ with $M[t\rangle M'$.
  It uses the same routing strategy than all other strategies. It routes a call if this call transfer does not cause a violation of the maximum thresholds of services.

Our goal in this evaluation is to calculate the reliability indicators according to SBP constraints, calls arrival and used strategy (strategies 1, 2, 3 and 4). The analysis of these indicators allows the evaluation of strategies' reliability in terms of processing time and proportion of lost calls. The evolution of the minimal calls processing time according to load variations is shown in Figure 4.9. The Figure 4.10 represents the evolution of the average calls processing time according to load variations. Figure 4.11 represents the evolution of the proportion of lost calls according to load variations. From these Figures we can differentiate two phases:

- We observe at the beginning of the execution (load: 0-3 calls) that the number of calls does not create an overload on services. In this case, we can see that the SBP reacts the same way with the four strategies (in terms of processing time and proportion of lost calls). This is explained by the fact that no duplication/consolidation is needed to handle the calls arrival. The four strategies are going to use only their routing strategy.

- We observe that from a load value greater than 3 calls (load: 4-15 calls) the number of calls creates an overload on the service $s2_1$ ($max_t(s2) = 3$). In this case, we can see that the process reacts in a different way according to the

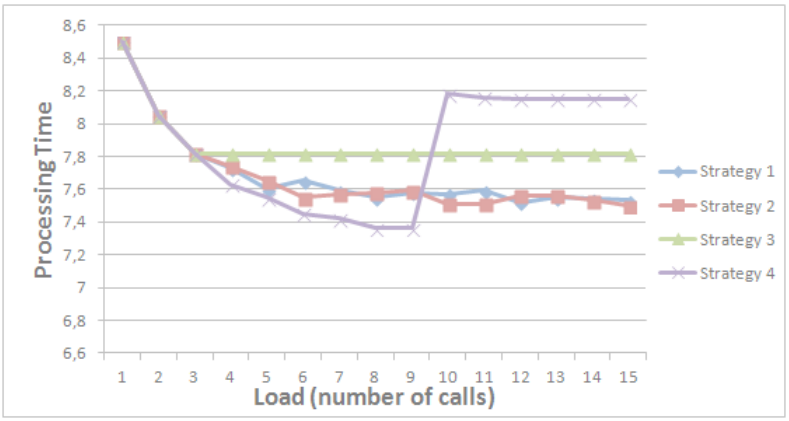


Figure 4.10: the evolution of the average calls processing time according to load variation

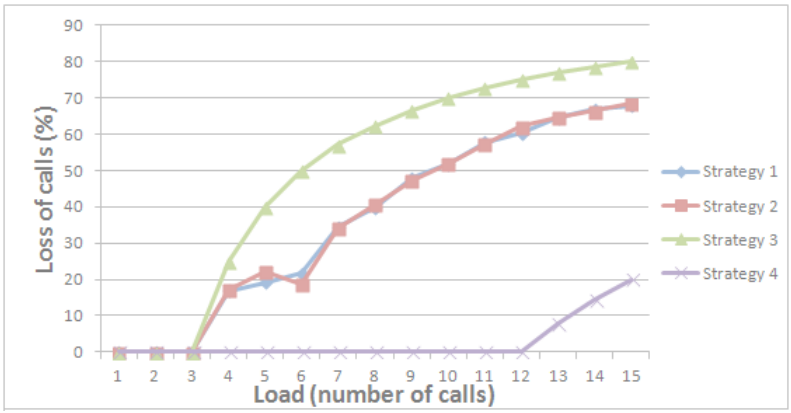


Figure 4.11: the evolution of the proportion of lost calls according to load variation

implemented strategy. In fact, the strategy 3 does not allow duplication of the overloaded service causing an important loss of calls. Strategies 1 and 2 react only when the service $s2_1$ becomes overloaded by duplicating it. We can see that these strategies are not suitable for ensuring the elasticity of SBPs with temporal constraints because of their late reactions. This delay in the reaction can cause a loss of calls. Strategy 4 prevents the overload of the service $s2_1$ by duplicating it when the load of the service $s1_1$ (the service that will transfer its calls to the service $s2_1$) may cause the overload of service $s2_1$ when its load will be transferred to service $s2_1$. We can see that this strategy is more suitable for ensuring the elasticity of the considered SBPs than the other strategies.

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

Hereafter, we give some possible executions of the SBP with the four strategies. These executions result from an initial marking $M_0 = 4'(s1_1, 0)$

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

- $T(x)$: action for the elapse of $x$ unit of time.
- $x'R(t)$: action that fires $x$ time the transition $t$ .
- $D(s)$: action that duplicates the service $s$.
- $C(s, s')$: action that consolidates the service $s'$ in its copy $s$.

**Strategy 3**

This strategy does not implement duplication/consolidation mechanisms. So, when the service $s2_1$ reached its maximum load threshold ($max_t(s2) = 3$) we cannot transfer a call to this service. It is necessary in this case to wait until the service is done with one of its current calls in order to receive a new one. Nevertheless, in the considered SBP if we wait until the service $s2_1$ finishes processing one of its calls and transfer it to another service ($s3_1$), the call waiting to be transferred to the service $s2_1$ will be expired (outdated call). Hereafter we give a possible execution of the SBP with strategy 3:

- $4'(s1_1, 0) \xrightarrow{T(2)} 4'(s1_1, 2)$
- $4'(s1_1, 2) \xrightarrow{3'R(t1_1)} 3'(s2_1, 0), 3'(s3_1, 0), (s1_1, 2)$
- $3'(s2_1, 0), 3'(s3_1, 0), (s1_1, 2) \xrightarrow{T(2)} 3'(s2_1, 2), 3'(s3_1, 2), \mathbf{(s1_1,4)}$
- $3'(s2_1, 2), 3'(s3_1, 2) \xrightarrow{3'R(t2_1)} 3'(s3_1, 0)$
- $3'(s3_1, 0) \xrightarrow{T(3)} 3'(s3_1, 3)$
- $3'(s3_1, 3) \xrightarrow{3'R(t3_1)} ()$

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

**Strategies 1 and 2**

Both strategies use duplication/consolidation mechanisms that react when a service is overloaded. In the considered SBP, when the service $s2_1$ reached its maximum load threshold ($max_t(s2) = 3$) both strategies duplicate it. The duplication of service $s2_1$ can generate two types of execution (favorable or unfavorable execution). The favorable execution solves the problem of overloaded service $s2_1$ while ensuring the treatment of all calls, when the unfavorable execution results in the loss of calls despite of the duplication of the overloaded service. Hereafter we give two possible executions (a favorable and an unfavorable one) of the SBP with the strategies 1 and 2. The favorable possible execution:

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

The unfavorable possible execution:

- $4'(s1_1, 0) \xrightarrow{T(3)} 4'(s1_1, 3)$
- $4'(s1_1, 3) \xrightarrow{3'R(t1_1)} 3'(s2_1, 0), 3'(s3_1, 0),\ (s1_1, 3)$

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

We notice that in the favorable execution we handle all SBP calls without any loss while in the unfavorable execution we lose a call (25% of loss). We observe that we have an average of $16,82\%$ loss with strategy 1 and $17,15\%$ loss with strategy 2.

**Strategy 4**

The strategy uses duplication/consolidation mechanisms that react when there is a possibility in the future, that a transfer of calls from a service to an another one creates an overload on the second service. In the considered SBP, before transferring calls to the service $s2_1$ the strategy duplicates the service $s2_1$ in order to have as copy as necessary of the service according to the load that it may receive from another service ($s1_1$). This strategy allows adaptation of the SBP in advance according to load that each service may receive in the future. Hereafter we give a possible execution with the strategy 4:

- $4'(s1_1,0) \xrightarrow{D(s2_1)} 4'(s1_1,1)$
- $4'(s1_1,1) \xrightarrow{T(2)} 4'(s1_1,3)$
- $4'(s1_1,3) \xrightarrow{3'R(t1_1)} 3'(s2_1,0), 3'(s3_1,0),\ (s1_1,3)$
- $3'(s2_1,0), 3'(s3_1,0), (s1_1,3) \xrightarrow{1'R(t1_2)} 3'(s2_1,0),\ (s2_2,0), 4'(s3_1,0)$
- $3'(s2_1,0), (s2_2,0), 4'(s3_1,0) \xrightarrow{T(3)} 3'(s2_1,3),\ (s2_2,3), 4'(s3_1,3)$
- $3'(s2_1,3), (s2_2,3), 4'(s3_1,3) \xrightarrow{3'R(t2_1)} (s2_2,3),\ (s3_1,3), 3'(s3_1,0)$
- $(s2_2,3), (s3_1,3), 3'(s3_1,0) \xrightarrow{1'R(t2_2)} 4'(s3_1,0)$
- $4'(s3_1,0) \xrightarrow{C(s2_2,s2_1)} 4'(s3_1,0)$

- $4'((s3_1, 0) \xrightarrow{T(3)} 4'(s3_1, 3)$
- $4'(s3_1, 3) \xrightarrow{4'R(t3_1)} ()$

## 4.5 Conclusion

Ensuring the effectiveness of elasticity requires using reliable strategies which are responsible on managing the execution of elasticity actions (duplication/consolidation). Since that many strategies can be used to ensure SBPs elasticity, it is necessary to evaluate and validate these strategies in order to ensure their effectiveness in providing SBPs elasticity before implementing them in real Cloud environments. In this chapter, we proposed to use the previously defined elasticity controller as a framework for the validation and evaluation of different elasticity strategies. Our evaluation approach intertwines two techniques: verification and simulation. On one hand, the verification allows formal verification of the correctness of elasticity strategies. On the other hand, the simulation allows comparing between different elasticity strategies in order to choose the most suitable one. For the Proof of Concept, we provided an example of the usage of the generic controller for strategies evaluation.

Chapter 5

# Provisioning of Elasticity Mechanisms in the Cloud

**Contents**


**5.1 Introduction** . . . . . **99**
**5.2 Provisioning Requirements** . . . . . **100**
**5.3 Provisioning of Elastic SBPs on Cloud infrastructures** . . . . . **102**
5.3.1 Service micro-containers . . . . . 102
5.3.2 Elastic Service micro-containers . . . . . 104
5.3.3 Experiment . . . . . 106
5.3.3.1 Use case . . . . . 106
5.3.3.2 Evaluation . . . . . 108
**5.4 Provisioning of Elastic SBPs on Cloud platforms** . . . . . **111**
5.4.1 Autonomic Management for OCCI Resources . . . . . 112
5.4.2 Autonomic Approach for SBPs elasticity . . . . . 115
5.4.2.1 Autonomic Infrastructure for SBPs elasticity . . . . . 115
5.4.2.2 Implementation . . . . . 116
5.4.3 Experiment . . . . . 119
5.4.3.1 Use Case . . . . . 119
5.4.3.2 Evaluation . . . . . 121
**5.5 Conclusion** . . . . . **123**


## 5.1 Introduction

Our work aim at proposing a novel approach for ensuring elasticity of SBPs in Cloud environments. We have proposed in chapter 3 a formal model for SBPs elasticity that features two elasticity operations (duplication/consolidation) and an elasticity controller that allows using different elasticity strategies. In chapter 4, we have proposed to use our elasticity controller as a framework for the evaluation of different elasticity strategies before implementing them in real Cloud environments.

In this chapter, we propose to go further by proposing an approach for the provisioning of our elasticity mechanisms in the Cloud. Our goal is to show how we can use the previously defined elasticity mechanisms to ensure SBPs elasticity in real Cloud environments without changing the nature of the SBPs or existing Cloud environments. Provisioning of elastic SBPs in Cloud environments can be done in two different contexts: Infrastructure (IaaS) context and platform (PaaS) context. In our work, we propose two approaches for the provisioning of SBPs elasticity while considering these two contexts.

We dedicate the first section of this chapter to describe the provisioning requirements for our elasticity approach. Afterwards, we present our approaches for the provisioning of elastic SBPs on Cloud infrastructures (section 5.3) and Cloud platforms (section 5.4).

## 5.2 Provisioning Requirements

In order to provision SBPs elasticity in the Cloud according to our elasticity approach (see Figure 5.1), we have to provide Cloud environments with mechanisms to meet the provisioning requirements. As showed previously, in our elasticity approach we need to:

- Operate at the scope of services that compose the SBP (i.e., duplicate/consolidate only the bottleneck services).
- Monitor these services to provide the elasticity controller with information about the SBP execution.
- Manage the duplication and consolidation of services (i.e., adding/removing copies, routing/balancing service calls).

Our goal here is to provision elastic SBPs in Cloud environments. This can be done in two different contexts (IaaS and PaaS):

- **Provisioning of Elastic SBPs in IaaS context:** In this context, only infrastructure resources are provided (e.g., VMs). In order to provision our elasticity approach for SBPs, we need to use service containers and/or process engines that allow operating at the scope of services. One solution consists in modifying existing containers and/or engines (e.g., apache ODE) to allow operating at the scope of services. Another solution consists in using the micro-container approach proposed in [63] [64] which is based on a simple idea that consists in dedicating a micro-container to each deployed service. Each micro-container can host and run its service and can be seen as a specialized container. This approach fits perfectly with an elasticity granularity that we consider to SBPs i.e. the scope of services. In addition, a monitoring extension for micro-containers

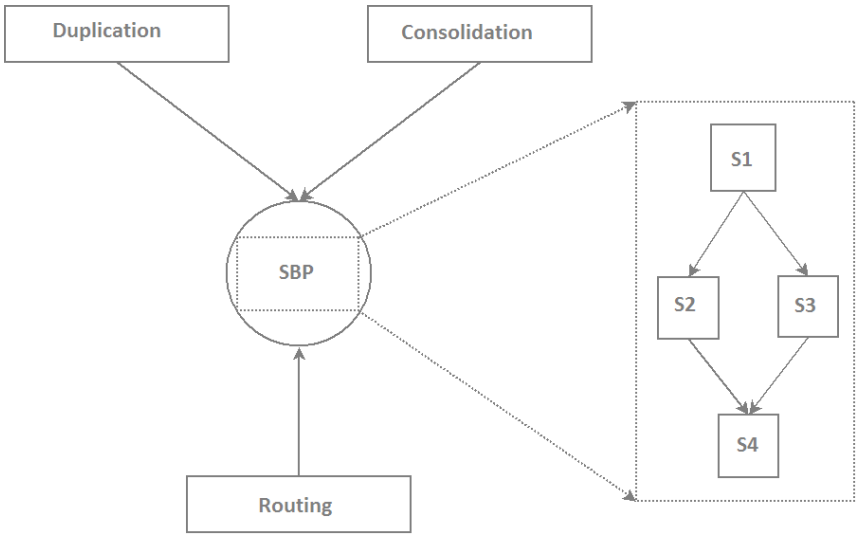


Figure 5.1: Elasticity approach for SBPs elasticity.

has been proposed in [65]. This extension provides mechanisms to collect, aggregate, and report monitoring data on each deployed micro-container. However, we still need to extend the micro-containers approach with mechanisms for enforcing SBPs elasticity (i.e., mechanisms of duplication/consolidation and an elasticity controller) on Cloud infrastructures.

- **Provisioning of Elastic SBPs in PaaS context:** In this context, we can take advantage of existing PaaS tools and mechanisms in order to provision elasticity mechanisms for SBPs. In fact, almost Cloud platforms provide tools and mechanisms for hosting and monitoring services as well as adding/removing services instances. However, we still need to set up an elasticity controller that monitors SBPs execution, analyzes monitoring data (using elasticity strategies) and executes elasticity actions (duplication/consolidation).

In the following, we propose two approaches for the provisioning of elastic SBPs in Cloud environments. The first approach packages non-elastic SBPs in micro-containers, extended with our elasticity mechanisms, before deploying them in Cloud infrastructures (IaaS). The second approach specializes an autonomic infrastructure for Cloud resources in order to dynamically add elasticity facilities to SBPs on Cloud platforms (PaaS).

## 5.3 Provisioning of Elastic SBPs on Cloud infrastructures

In this section, we present our approach for the provisioning of elastic SBPs on Cloud infrastructures. Provisioning of our elasticity mechanisms requires operating at the scope of services. As stated before, this can be done using the micro-container approach for service provisioning in the Cloud [63] [64]. This approach fits perfectly with an elasticity granularity that we consider to SBPs i.e. the scope of services. Therefore, we propose in our work to extend the micro-containers approach, using our elasticity mechanisms, in order to provision elastic SBPs on Cloud infrastructures.

We introduce the already performed service micro-containers and their correspondent packaging framework in Section 5.3.1. Then, we present and discuss in Section 5.3.2 our approach to extend the packaging framework in order to add elasticity facilities to micro-containers. In Section 5.3.3, we experiment our proposal in a real Cloud infrastructure.

### 5.3.1 Service micro-containers

In [63] [64], authors demonstrated that the use of classical service containers (e.g., Apache Tuscany, Apache ODE, Apache Axis, etc.) is not suitable for Cloud environments. In fact, they highlighted that classical service containers are neither scalable nor elastic. To address these drawbacks, they proposed novel service containers called micro-containers. This approach is based on a simple idea that consists in dedicating a micro-container to each deployed service in Cloud environments. Each micro-container can host and run its service and can be seen as a specialized container. Micro-containers are automatically generated and provide minimal and personalized facilities to manage the life cycle of deployed services (e.g., services hosting, interaction with clients, etc.). For example, the micro-container approach does not have a management competitions module as the approach assumes a single service per container.

The design of the micro-container approach is composed of two main parts: (1) the service micro-container, and (2) the generic packaging platform that builds the micro container that contains the service to be deployed. An overview of the main components of the packaging framework and architecture of generated micro-containers are detailed in Figure 5.2.

To package a service and build the appropriate micro-container for it, one must mainly provide for the deployment framework two elements:

1. The service to package with all its components (code, resources, etc.);
2. A deployment descriptor that specifies the container options.

The *Processor module* analyzes the deployment descriptor, process the service source code, detects the service binding types, instantiates an appropriate *Com-*

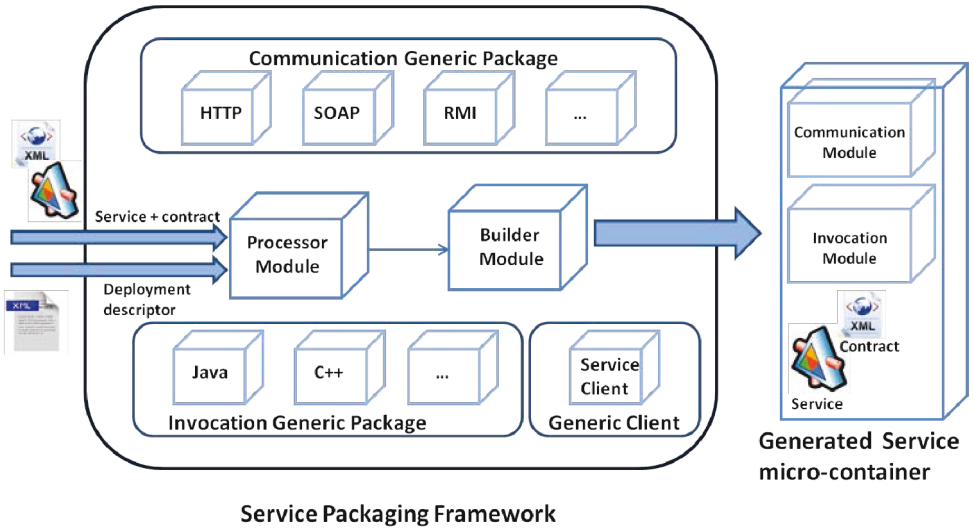


Figure 5.2: Service micro-container packaging framework.

*munication module* implementing these bindings from the *Communication Generic Package* and associates it to the service sources. The same principle is also followed for the selection of the *invocation module* to run the service once packaged in the micro-container. Based on the service implementation programming language, the appropriate *invocation module* is instantiated from the *Invocation Generic Package*.

The resulting code represents the generated micro-container code. It is composed only of the necessary modules for the deployed service, no more, no less. Generated micro-container hosts the service and implements its bindings regardless its communication protocol support as long as they are included in the *Generic Communication Package* and regardless the programming language as long as they are included in the *Invocation Generic Package*. Adding new communication protocols or programming languages support consists in adding the correspondent components in the packaging framework generic packages.

Henceforth, each generated service micro-container consists at least of three modules:

- A *Communication module* to establish communication and to support connection protocol.
- An *Invocation module* to process ingoing and outgoing data into and out of the server (packing and unpacking data).
- A Service module to store and invoke the packaged service and its contract (service descriptor).

The framework provides also a generic client to invoke packaged service in a generated micro-container. The client setup is based on the service bindings type and

information described in the contract. For example, for a Web service the contract is a WSDL document [66] and needed informations to setup the client are described in *operation*, *input*, *output* and *service* elements.

The design of the micro-container was made so that other modules can be added as extensions or add-on to support other non-functional properties (e.g., See [67] for mobility features support, See [65] for monitoring facilities support). We propose in our work to extend the micro-container approach in order to support SBPs elasticity on Cloud infrastructures.

### 5.3.2 Elastic Service micro-containers

Using micro-containers to host services corresponds perfectly to the need of addressing SBPs elasticity in a fine-grained manner. Indeed, they allow operating at the scope of services by deploying each service of the SBP in its specialized micro-container. As it is, the service micro-container packaging framework can be extended with elasticity faculties to support SBPs elasticity. To this end, we propose in this work to extend this framework by adding an elasticity facilities package that implements our previously defined elasticity mechanisms.

An overview of the extended deployment framework is described in Figure 5.3. This extension consists in adding two modules:

- A *Generic router* module to manage the duplication/consolidation of services (variation of the copies number).
- A *Generic controller* to monitor, analyze monitoring data and execute elasticity actions (duplication/consolidation).

The input of the extended framework is the deployment descriptor of the SBP. This latter is used to extract information about the services that compose the SBP as well as the interactions between these services. The deployment framework generates a service micro-container for each service of the SBP (i.e., as many micro-containers as services).

Our previously defined elasticity approach ensures SBPs elasticity by adding/removing services copies (i.e., service duplication/consolidation). To manage these services copies and route invocations to them, we propose to generate a router component for each service of the SBP. Each router is considered as a service and hence deployed within a dedicated micro-container. The router is responsible for managing the variation in the copies number of its considered service. It is also responsible of balancing calls between these service copies (i.e., determining under which condition a call is transferred over the set of service copies) according to its routing strategy (e.g., Round Robin, Random, etc.). Each router has a routing table which contains information about the different copies of its related service. Since initially a deployed service has one copy, the routing table has one entry to that copy. This routing table

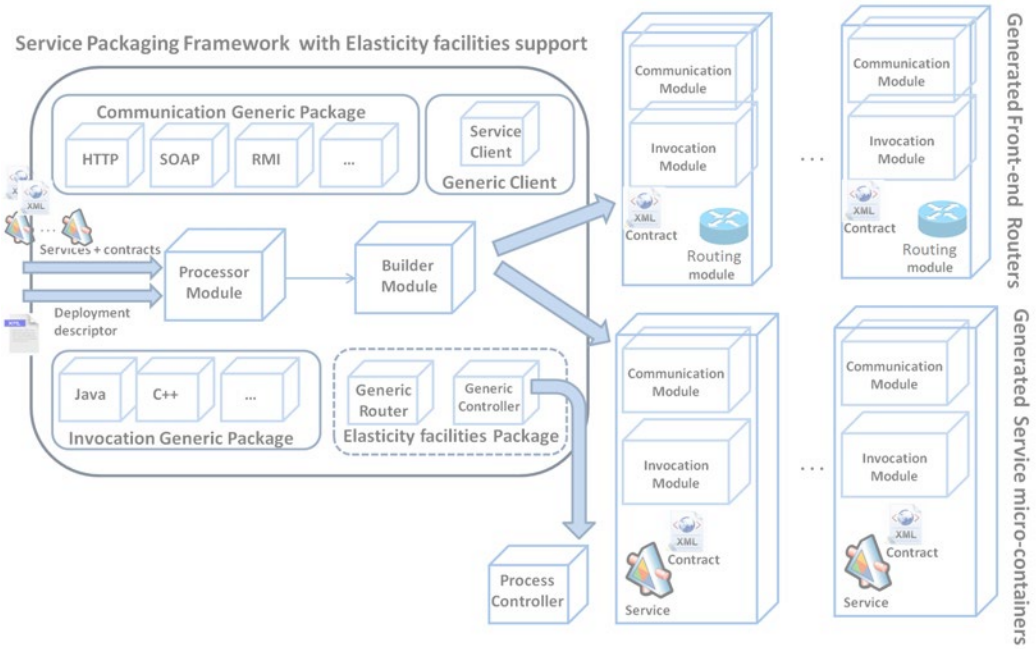


Figure 5.3: Extended service micro-container packaging framework.

is updated after each elasticity action (add a new service entry in case of duplication or remove an old service entry in case of consolidation).

Once the SBP services and their associated routers generated, we need to monitor them in order to provide the elasticity controller with information about their executions. To this end, we use the micro-container extension for monitoring proposed in [65]. This monitoring extension provides the mechanisms to collect, aggregate, and report monitoring data related to each service of the SBP. The metrics to monitor are extracted from the services contracts. These collected data will be used by the elasticity controller to decide on executing elasticity actions (duplication/consolidation).

The last step of our provisioning approach consists in the generation of the controller to be deployed as a service within a micro-container. It receives and analyzes the data according to the implemented elasticity strategy. The controller implements two elasticity actions (duplication/consolidation) that are performed after verifying the elasticity conditions ($ready_D$ for duplication and $ready_C$ for consolidation). Note here that we can assign a single controller for all the deployed processes, a controller for each subset or even a controller for each deployed process. We point out that the choice of the best deployment scenario needs a more thorough study that we consider in our future work. Consequently, to alleviate the burden, we propose in this work to use one elasticity controller for each deployed SBP.

At this step, we presented the different steps that allow the provisioning of elastic

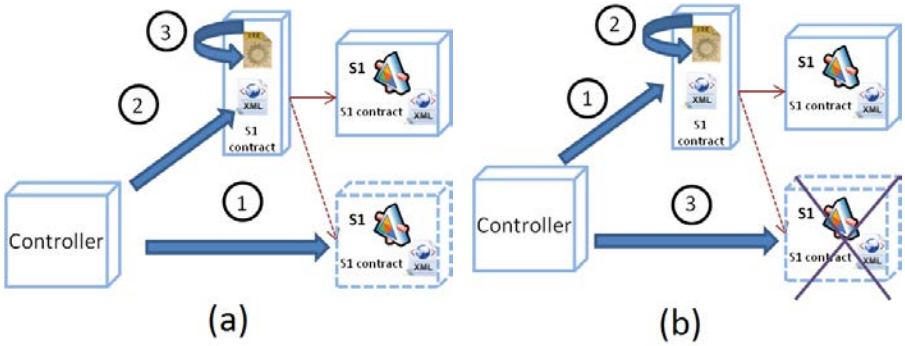


Figure 5.4: Service micro-containers (a) duplication and consolidation (b) steps.

SBPs on Cloud infrastructures. Figure 5.4 shows how duplication/consolidation actions are performed. A first use of these duplication/consolidation actions is when a service of the SBP is overloaded. The controller checks whenever the duplication condition ($ready_D$) is verified or not. If the condition is verified, the controller executes a duplication action to add a new copy of the concerned service (Figure 5.4-(a),action 1). Then, the controller updates the router by adding the new entry to the routing table (Figure 5.4-(a), actions 2 and 3). The second use case is when a service is underloaded. If the consolidation condition ($ready_C$) is verified, the controller applies a consolidation action to remove the concerned copy of service from the routing table (Figure 5.4-(b), actions 1 and 2). Then, the controller removes the concerned copy of service (Figure 5.4-(b), action 3).

### 5.3.3 Experiment

In this section, we present and comment a use case scenario to show how we ensure SBPs elasticity in a Cloud infrastructure according to our previously defined elasticity approach. We start by detailing the use case that we realized. Then we present the experiments scenarios and their results.

#### 5.3.3.1 Use case

To realize our use case, we propose to ensure the elasticity of the previously defined SBP of the online computer shopping service presented in Figure 3.1. This SBP is composed of four elementary services that interact to ensure the business functionality of the process. The interactions between these services are described in a PNML document [68].

According to our extension detailed in Section 5.3.2, the packaging framework execution induces the generation of a set of resources: four micro-containers to host the SBP services, four routers to be deployed upstream the micro-containers and an

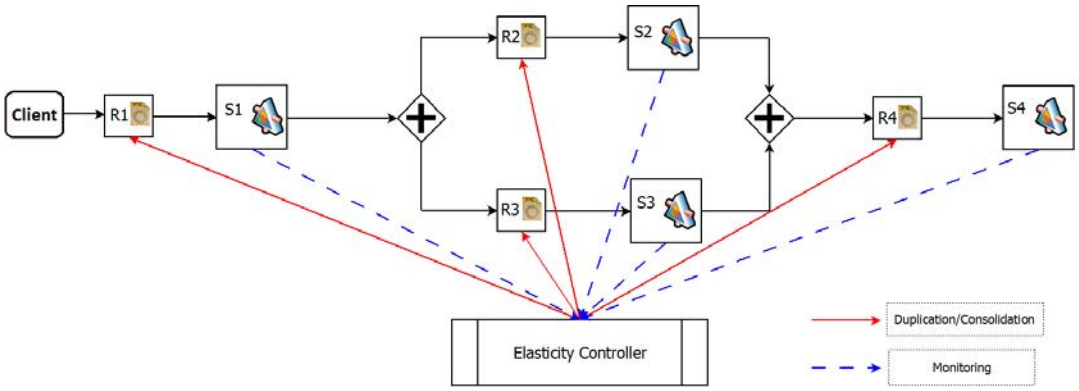


Figure 5.5: An example of the provisioning of an elastic SBP

elasticity controller micro-container which analyzes the monitored data and performs elasticity actions. Concretely, the obtained execution chain is presented in Figure 5.5. The execution of this chain is equivalent to the execution of the initial process (Figure 3.1). Preserving the semantic functionality of the process is guaranteed by our duplication/consolidation operations.

These micro-containers (i.e., services, routers and a controller) can be deployed as standalone applications in one or several virtual machines provided by an IaaS manager such as OpenNebula [69] or Openstack [70].

We propose in this use case to use services response times as a metric (scaling indicator) to base our elasticity strategies. Consequently, each service micro-container sends the response time of its hosted service to the controller. Afterwards, the controller analyzes the received response times and decides to trigger an elasticity action or not according to the implemented strategy. In order to illustrate our approach, we propose to instantiate the controller with two elasticity strategies (strategy 1 and strategy 2). These strategies trigger a duplication action if the response time is over the max threshold ($max_t$). In the other case, these strategies trigger a consolidation action if the response time is under the min threshold ($min_t$). Hereafter the thresholds used for each strategy:

**Strategy 1**

- Max_t($s1$) = 5 ms. Max_t($s2$) = 10 ms. Max_t($s3$) = 5 ms. Max_t($s4$) = 5 ms.
- Min_t($s1$) = 2 ms. Min_t($s2$) = 5 ms. Min_t($s3$) = 1 ms. Min_t($s4$) = 1 ms.

**Strategy 2**

- Max_t($s1$) = 5 ms. Max_t($s2$) = 75 ms. Max_t($s3$) = 5 ms. Max_t($s4$) = 5 ms.
- Min_t($s1$) = 2 ms. Min_t($s2$) = 20 ms. Min_t($s3$) = 1 ms. Min_t($s4$) = 2 ms.

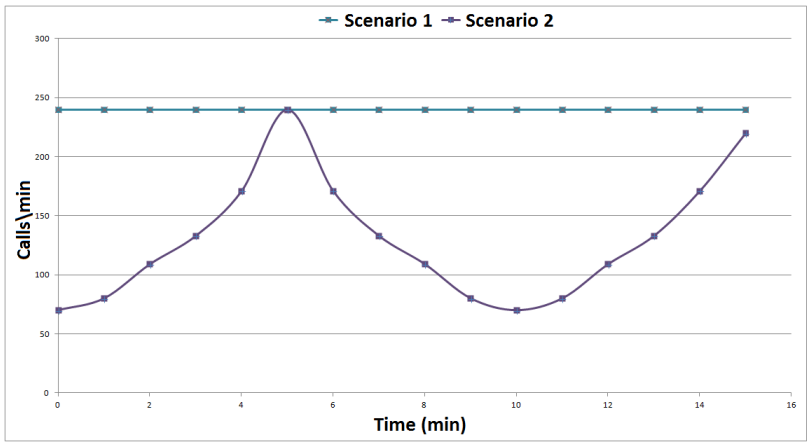


Figure 5.6: Calls arrival scenarios for invoking the SBP

In our use case we choose two calls arrival scenarios to invoke the SBP (See Figure 5.6). In the scenario 1, the calls arrival is constant (240 calls/min). In the scenario 2, the calls arrival is variable (between 60 calls/min and 240 calls/min). The client that we used to invoke the SBP is a REST client that sends requests to the endpoint of the SBP.

#### 5.3.3.2 Evaluation

To perform our evaluations we used the NCF (Network and Cloud Federation) experimental platform deployed at Telecom SudParis, France. The NCF experimental platform aims at merging networks and Cloud concepts, technologies and architectures into one common system. NCF users can acquire virtual resources to deploy and validate their own solutions and architectures. The network is in constant evolution and has for information: 380 Cores Intel Xeon Nehalem, 1.17 TB RAM and 100 TB as shared storage. Two Cloud managers allow managing this infrastructure and virtual resources i.e., OpenNebula [69] and OpenStack [70]. In our case, we used OpenNebula which is a virtual infrastructure engine that provides the needed functionality to deploy and manage virtual machines (VMs) on a pool of distributed physical resources. To create a VM, we can use one of the three predefined templates offered by OpenNebula i.e. SMALL, MEDIUM and LARGE, or we can specify a new template. During our experiments, we used our specific template with the following characteristics: 4 cores (2.66 GHZ each core) and 4 Gigabytes of RAM.

In our evaluations, we deploy a SBP as standalone applications (using micro-containers) in a virtual machine provided by OpenNebula. Our goal is to show how we can provision elastic SBPs, according to our elasticity approach, in order to maintain their QoS.

To show the efficiency of our solution, we will answer to the following questions:

- How does the elasticity strategies enhance the response time of the SBP?
- How does the elasticity strategies influence the resource consumption of the SBP?

We calculate the response time of the SBP before and after adding elasticity. When we added elasticity, we used two elasticity strategies in order to evaluate the influence that a strategy has on response time and resource consumption. To do this, we deployed the SBP and used the REST Client to call it according to the calls arrival scenarios defined previously (Figure 5.6). The experimental results with the scenario 1 are presented in Figures 5.7 and 5.8. The experimental results with the scenario 2 are presented in Figures 5.9 and 5.10.

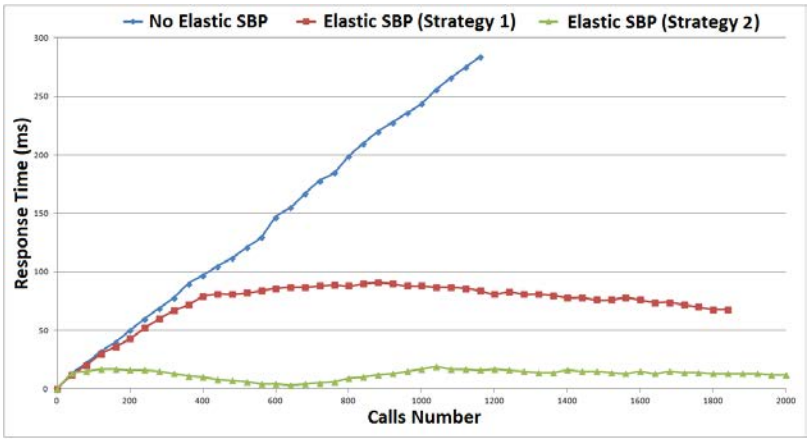


Figure 5.7: Response time of the SBP before and after adding elasticity with the scenario 1.

The analysis of the experimental results shows that the response time of the SBP without our mechanisms is increasing proportionally with the number of clients calls. Moreover, there is a loss of calls (42% in the case of scenario 1 and 8% in the case of scenario 2) which is the result of the overload of the SBP. In contrast, using our mechanisms, whenever the response time reached the max threshold of a service (i.e., 75 ms in the case of strategy 1 and 10 ms in the case of strategy 2 for the service $s2$) an elasticity action is triggered to duplicate the overloaded service by adding a new instance to it. This elasticity action decreases the response time of all the SBP. If the response time is under the min threshold (i.e., 20 ms in the case of strategy 1 and 5 ms in the case of strategy 2 for the service $s2$), an elasticity action is triggered to consolidate the service instances. The experiments show the efficiency of our approach to enhance the behavior of the SBP. In fact, the response time remains around the

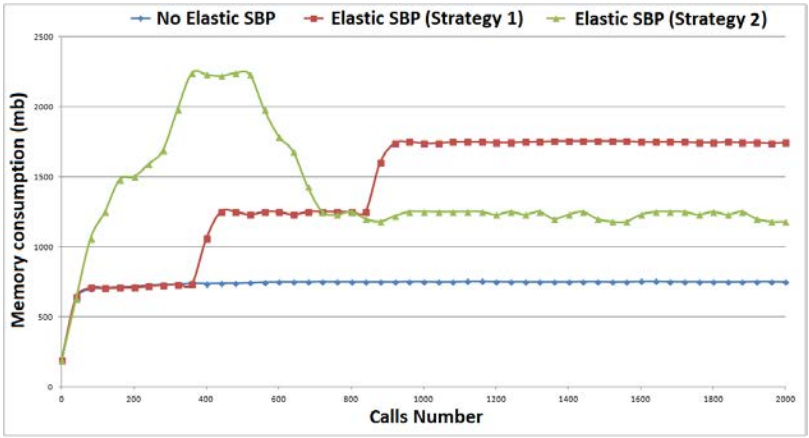


Figure 5.8: Memory consumption of the SBP before and after adding elasticity with the scenario 1.

specified thresholds. The experiments show also that with the strategy 2, we reached 2000 clients calls without any loss of calls. As shown in the experimental results, in some points, the response time is over the max threshold, this is due to the number of clients calls sent before the elasticity action takes effect.

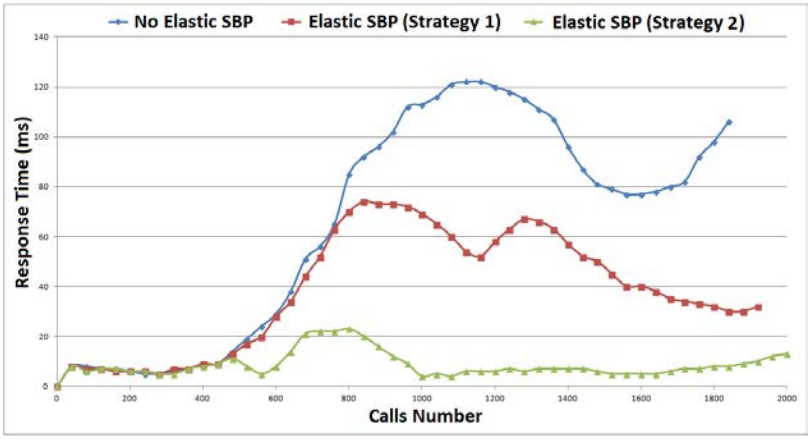


Figure 5.9: Response time of the SBP before and after adding elasticity with the scenario 2.

The experimental results show that both strategies enhance the QoS of the SBP (in terms of response time). However, using the strategy 1, we still have loss of calls (8% in the case of scenario 1 and 4% in the case of scenario 2). In contrast, using the strategy 2, the SBP treats all calls without any loss of calls. This difference is explained by

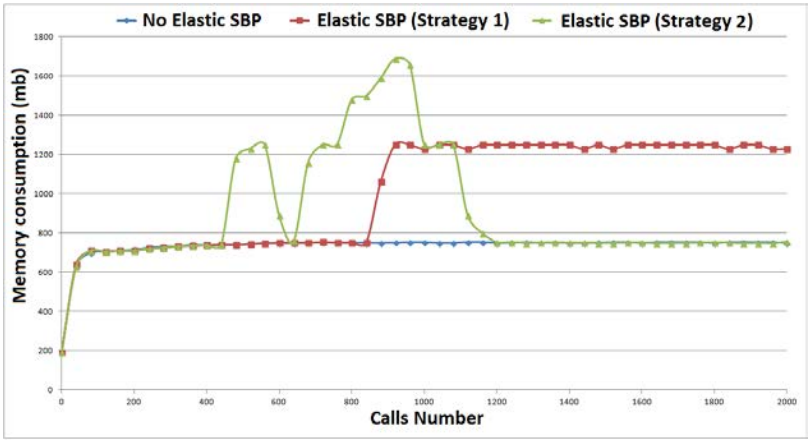


Figure 5.10: Memory consumption of the SBP before and after adding elasticity with the scenario 2.

the difference on thresholds used in the two strategies. Indeed, the strategy 2 reacts faster to the increase of the response time than strategy 1 which allows the treatment of all calls, while the strategy 1 reacts late which cause a loss of calls. Based on the results, the strategy 2 ensures a better efficiency and allows maintaining the QoS of the deployed SBP in this use case. Over and above the response time of the SBP, we measured the memory consumption of our SBP. The results shown in Figures 5.8 and 5.10 shows the difference between the memory consumption of a non-elastic SBP and an elastic one (with strategy 1 and strategy 2). The memory consumption of a non-elastic SBP is constant since each service has its allocated memory and no new instances are added. Meanwhile, for an elastic SBP, the curve shows that the memory consumption is changing whenever there is duplication or consolidation.

In this section, we proposed an end-to-end approach for the provisioning of elastic SBPs in real Cloud infrastructures. Based on the already performed service micro-container approach, we proposed to extend the packaging framework by adding new generic modules implementing our elasticity mechanisms. We showed that our approach is efficient when applied to a real SBP deployed in a real Cloud infrastructure. The obtained results show that we ensure the elasticity of deployed SBPs.

## 5.4 Provisioning of Elastic SBPs on Cloud platforms

In this section, we propose our approach for the provisioning of elastic SBPs on Cloud platforms. To this end, we can use and take advantage of existing PaaS tools and mechanisms (e.g., monitoring components, mechanisms for adding/removing instances) to ensure SBPs elasticity. In order to provision elasticity on Cloud platforms, we need to set up an elasticity controller to analyze monitoring data and manage elas-

ticity mechanisms. To do this, we propose in our work to adapt and specialize the autonomic infrastructure for Cloud resources proposed in [71] in order to ensure SBPs elasticity on Cloud platforms [72].

We will start by introducing the proposed approach for autonomic management of Cloud resources (Section 5.4.1). Afterwards, we propose to specialize an autonomic infrastructure, based on the previously defined elasticity approach, to dynamically add elasticity facilities to SBPs (Section 5.4.2). Finally, we will use the defined concepts to realize a use case to experiment our autonomic infrastructure for SBPs elasticity in a real Cloud platform (Section 5.4.3).

### 5.4.1 Autonomic Management for OCCI Resources

IBM [53] defines Autonomic Computing as the ability to manage computing resources that automatically and dynamically respond to the requirements of the business based on SLA. Autonomic management is usually presented as a Monitor, Analyze, Plane and Execute (MAPE) loop [73, 74]. Autonomic management features for Cloud resources are presented in Figure 5.11. In this autonomic loop, the central element represents any Cloud resource for which we want to exhibit an autonomic behavior.

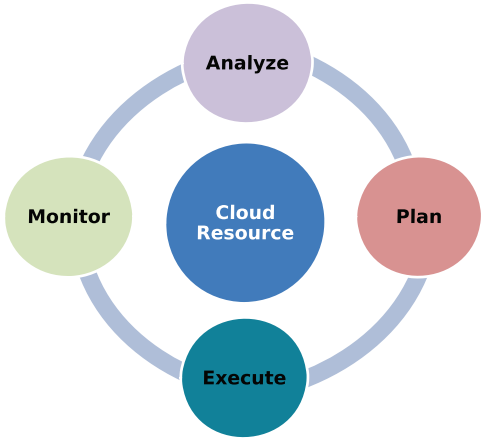


Figure 5.11: Autonomic control loop for a Cloud resource

#### Adding Autonomic Management for Cloud Resources

To add autonomic management features for Cloud resources, Mohamed et al. [71] proposed an approach that dynamically adds the needed resources and extensions to ensure the MAPE (Monitor, Analyze, Plan and Execute) functions.

In their approach, they proposed to extend the resource by adding the needed mechanisms that allow to retrieve data from the resource and to execute reconfig-

urations on it. The extensions consist in adding new functions that transforms the resource into a managed Resource. The Monitoring function is ensured by adding two extensions: (1) Polling extension that defines monitoring by polling functionality, and (2) Subscription extension that defines monitoring by subscription functionality as defined for a publish/subscribe system. In order to ensure the Execute function, a Reconfiguration extension is added to allow applying reconfiguration actions on the extended resource.

For the *Analyze* and *Plan* functions, they defined two new abstract resources responsible of these functions. To support the *Analyze* function they define the *Analyzer* resource that allows analyzing monitoring data and eventually generating alerts whenever the QoS is and/or will not be respected. Moreover, in order to support the Plan function, they define the *Planner* resource that receives alerts from the *Analyzer*. Based on these alerts, the *Planner* may generate reconfiguration actions to be applied on Resources. The customization of these abstract resources consists on specifying the strategies to be used by the *Analyzer* to process the incoming monitoring information. These strategies are extracted from the SLA of the managed Resource. The customization of the *Planner* is based on specifying the actions to be used for the processing of the incoming alerts.

To be widely adapted among Cloud providers, the proposed approach was built upon the Open Cloud Computing Interface (OCCI) standard.

### OCCI extension for Autonomic Management

The Open Grid Forum defines the Open Cloud Computing Interface (OCCI) [75] as "*an abstraction of real world resources, including the means to identify, classify, associate and extend those resources.*" OCCI provides a model to represent resources and an API to manage them. OCCI specifications consist essentially of three documents. The first is OCCI Core [75] that formally describes the OCCI Core Model. As shown in Figure 5.12, OCCI Core describes the Cloud resources as instances of the class *Entity* which can be a *Resource* or a *Link*. Each *Entity* instance could be extended using one or more instances of the class *Mixin*. The Mixin mechanism allows new Resources capabilities to be added at instantiation time or at runtime. The second document is OCCI HTTP Rendering [76] that describes how to interact with OCCI Core using HTTP protocol. The third document is OCCI Infrastructure [77] that is an extension of the Core Model to represent the Cloud infrastructure layer. The OCCI model can be extended to cover different layers in the Cloud (i.e., IaaS [77], PaaS [78], SaaS [79], etc.).

In order to provide all the mechanisms to establish an autonomic loop for Cloud resources, an OCCI extension was proposed as shown in Figure 5.12. To dynamically establish this loop, an *Autonomic Manager* is defined as a generic OCCI Resource. It inspects a given SLA and carries out the list of actions to build the autonomic computing infrastructure. From a given SLA, this Resource determines monitoring

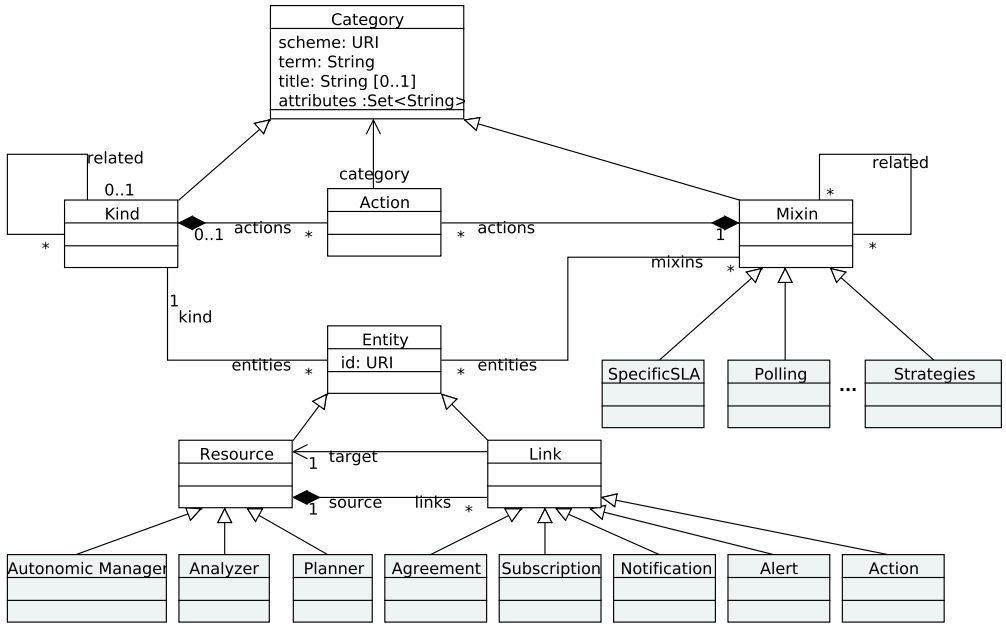


Figure 5.12: OCCI extension for Autonomic Computing.

targets (i.e., the needed metrics to be monitored). It is also responsible of extracting the strategies to be used by the *Analyzer* Resource and the reconfiguration actions to be used by the *Planner*. After inspecting the contract (SLA), the Autonomic Manager instantiates the needed Entities (i.e., Resources and Links). Then, it customizes these Entities with the needed Mixins and eventually configures them with the needed parameters.

The *Autonomic Manager* is responsible of extending a given resource to add the Monitor and Execute functions. For this aim, three Mixins are defined to transform any Resource to a managed Resource: (1) *Polling* Mixin describes the needed operations to monitor a Resource by polling; (2) *Subscription* Mixin to manage subscriptions on specific metrics in order to send notifications containing monitoring data; and (3) The *Reconfiguration* Mixin that provides the needed functionalities to ensure reconfiguration.

The *Analyzer* Resource is defined as a generic OCCI Resource that allows to analyze monitoring data and eventually to generate alerts. The *Analyzer* Resource could be specified using *Strategies* Mixin collection. This Mixin implements a computation strategy that triggers specific alerts based on incoming monitoring information. It represents a function applied by the *Analyzer* to compare monitoring information values against previously defined conditions. Whenever a condition is not respected, the Mixin instance makes the *Analyzer* trigger a specific alert. A Subscription is specified

with a Mixin instance from the *Subscription Tool* Mixin collection. This Mixin may describe the mechanism of the subscription (using HTTP, SMS, or email, etc.) or may refer to an external program that handles this task. The *Analyzer* processes the incoming monitoring notifications and eventually generates alerts whenever the SLA is violated. The generated alerts are sent to the *Planner*. The *Planner* is a generic OCCI Resource that receives alerts from the *Analyzer*. This Resource is an abstract description that uses instances of *Reconfiguration Actions* Mixin collection that implements a computation action to be applied on incoming alerts, the *Planner* triggers reconfiguration actions on specific Resources using instances of this Mixin. It can also refer to an external program that handles planning for reconfigurations.

This approach provides an on demand and generic autonomic infrastructure for Cloud resources in Cloud environments. In our work, we propose to adopt and adapt this autonomic infrastructure in order to provide SBPs elasticity facilities on Cloud platforms.

### 5.4.2 Autonomic Approach for SBPs elasticity

In this section, we propose to specialize the previously defined autonomic infrastructure for Cloud resources in order to ensure the elasticity of SBPs on Cloud platforms. To do this, we will start by presenting a general view of the usage of the previously defined OCCI Entities (i.e., Resources and Links) and Mixins to establish our autonomic computing infrastructure for SBPs elasticity. Afterwards, we detail the implementation aspects of our proposal.

#### 5.4.2.1 Autonomic Infrastructure for SBPs elasticity

We propose to specialize the autonomic infrastructure, based on our previously defined elasticity approach, to dynamically add elasticity facilities to SBPs (see Figure 5.13). The elasticity controller will cover the Analyze and Plan functions of the MAPE loop in order to ensure SBPs elasticity. To do this, we merely break our elasticity controller previously defined between the *Analyzer* and the *Planner* Resources. Consequently, the elasticity conditions (*ready_D* and *ready_C*) are implemented as instances of the *Strategies* Mixin while the elasticity actions (duplication and consolidation) are implemented as instances of the *Reconfiguration Actions* Mixin. Notwithstanding, spurred by the fact that almost all the existing PaaS providers offer the routing functionality, we delegate the routing mechanisms to the targeted environment in which we will deploy our SBP.

To establish our autonomic computing infrastructure, we start by setting up our OCCI Server. This server is responsible of instantiating any OCCI Entity. The first Resource instantiated in this server is the *Autonomic Manager* Resource. Based on the SLA, the *Autonomic Manager* detects the attributes or services that need to be monitored for the managed Resource (i.e., SBP). Consequently, it extends the managed Resource by monitoring Mixins (i.e., Polling and/or Subscription). Moreover, it

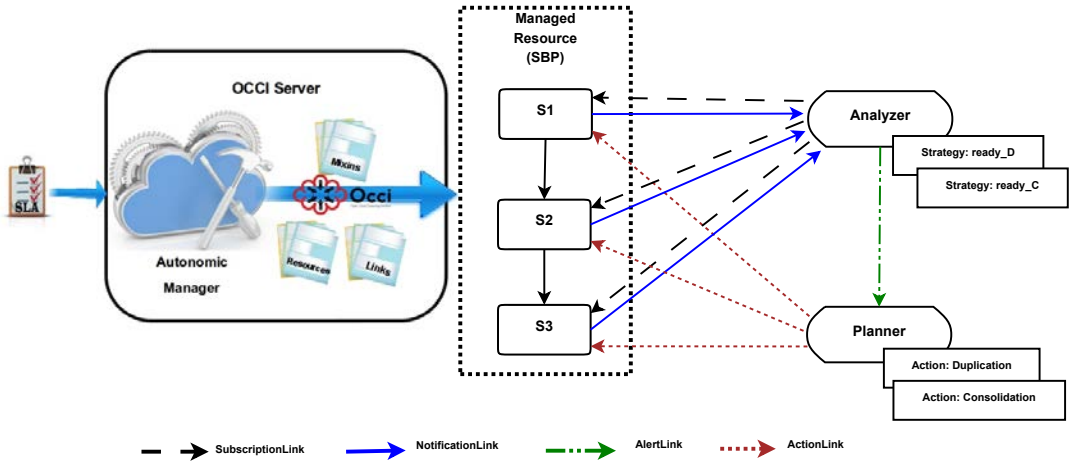


Figure 5.13: Autonomic Computing Infrastructure establishment for SBP elasticity.

extends it by a *Reconfiguration* Mixin to enable reconfigurations (i.e., service duplication/consolidation). The *Autonomic Manager* sends a request to the OCCI Server to instantiate the Mixed OCCI Resource (i.e., the basic Resource with their newly added Mixins). Whenever the managed Resource is ready, the *Autonomic Manager* orders the Server to deploy and start it.

The next step realized by the *Autonomic Manager* is to instantiate and customize the needed Resources and Links in order to establish the autonomic infrastructure. For ease of presentation, we refrain from presenting all the Mixins in Figure 5.13. Therefore, we kept just the needed Mixins instances of *Strategies* Mixin (i.e., elasticity strategies) and *Reconfiguration Actions* Mixin (i.e., elasticity actions). In fact elasticity strategies could be described as instances of *Strategies* Mixin (see Figure 5.12). In addition, elasticity actions could be described as instances of the *Reconfiguration Actions* Mixin (see Figure 5.12).

The *Autonomic Manager* instantiates the *Analyzer* and subscribes it to the managed Resource. At the reception of a notification, the *Analyzer* uses *Strategies* Mixin to process incoming monitoring information. If one of the strategies is verified, the *Analyzer* may raise alerts to the *Planner*. Accordingly, the *Autonomic Manager* instantiates the *Planner* and links it to the *Analyzer*. The *Planner* generates a plan for reconfiguration actions. As shown in Figure 5.13, the used plans are responsible of generating elasticity actions. The last step is to link the *Planner* to the managed Resource in order to use the generated reconfiguration actions and apply them on the managed Resource.

#### 5.4.2.2 Implementation

To implement the different Resources, Links and Mixins that we previously defined we used a JAVA implementation provided by OCCI working group called occi4java

[80]. This implementation is developed by the OCCI working group according to the specifications. The project is divided in three parts. Each of these parts corresponds to one of the three basic documents of OCCI specifications (i.e., OCCI Core, OCCI Infrastructure and OCCI HTTP Rendering). Using occi4java, we extended the Resource class to create our own resources for autonomic computing. We also extended the Link class to define our links.

Since the Mixin mechanism is not a native functionality for JAVA, we used the mixin4j framework [81]. This framework allows creating java Mixins by the use of annotations. We used this framework to implement the different Mixins containing the needed functionalities for our autonomic infrastructure.

For REST HTTP Rendering, the occi4java project uses Restlet framework [82]. Restlet is an easy framework that allows adding REST mechanisms. After adding the needed libraries, one needs just to add the Restlet annotations to implement the different REST actions (i.e., POST, GET, PUT and DELETE). Then to set up the server, one has to create one or more Restlet Components and attach the resources to them. We used the proposed annotations to define the needed actions for our OCCI Resources, Links and Mixins. That allowed us to use the HTTP REST Rendering to manage our autonomic infrastructure. And in order to enforce the scalability of our solution, we implemented different Restlet Components to allow the distribution of our infrastructure.

Finally, we used WS-Agreement to describe the SLA (see Listing 5.1). WS-Agreement is an extensible language to describe SLAs. The extendability of this language allows defining user specific elements and using them in the SLA. In our example, we were able to specify the attribute that needs to be monitored and their different thresholds. We added new elements to describe the analysis strategies (i.e., *<strategy:Ready_D>*) and the reconfiguration actions (i.e, *<plan:ReconfigurationAction>*).

To realize our use case, we implemented different monitoring Mixins. The first one uses the REST API proposed by NewRelic [83] service integrated in CloudFoundry PaaS. This Mixin gets the needed information about the application and creates notifications related to the monitored attributes. The second one consists on applying a transformation on the byte-code of the application to make it send notifications about its execution time [84].

Listing 5.1: WS-Agreement sample

```
1<?xml version="1.0" encoding="UTF-8"?>
2<wsag:AgreementOffer AgreementId="ab97c409">
3 <wsag:Name>xs:CFApplicationAg</wsag:Name>
4 <wsag:AgreementContext>
5   ...
6 </wsag:AgreementContext>
7 <wsag:Terms>
8  <wsag:All>
9   <wsag:GuaranteeTerm wsag:Name="g1" wsag:Obligated="ServiceProvider">
10    <wsag:ServiceLevelObjective>
```

```
11      <wsag:KPITarget>
12       <wsag:KPIName>time_response_maxthresholds</wsag:KPIName>
13       <wsag:CustomServiceLevel
14        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
15        xmlns:exp="http://www.telecom-sudparis.com/exp">
16        <exp:Less>
17         <exp:Variable>S1_maxthreshold</exp:Variable>
18         <exp:Value>100ms</exp:Value>
19         <exp:Variable>S2_maxthreshold</exp:Variable>
20         <exp:Value>1000ms</exp:Value>
21         <exp:Variable>S3_maxthreshold</exp:Variable>
22         <exp:Value>100ms</exp:Value>
23         <exp:Variable>S4_maxthreshold</exp:Variable>
24         <exp:Value>100ms</exp:Value>
25          ...
26        </exp:Less>
27       </wsag:CustomServiceLevel>
28       <wsag:KPIName>time_response_minthresholds</wsag:KPIName>
29       <wsag:CustomServiceLevel
30        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
31        xmlns:exp="http://www.telecom-sudparis.com/exp">
32        <exp:Greater>
33         <exp:Variable>S1_minthreshold</exp:Variable>
34         <exp:Value>10ms</exp:Value>
35         <exp:Variable>S2_minthreshold</exp:Variable>
36         <exp:Value>10ms</exp:Value>
37         <exp:Variable>S3_minthreshold</exp:Variable>
38         <exp:Value>500ms</exp:Value>
39         <exp:Variable>S4_minthreshold</exp:Variable>
40         <exp:Value>10ms</exp:Value>
41          ...
42        </exp:Greater>
43       </wsag:CustomServiceLevel>
44      </wsag:KPITarget>
45     </wsag:ServiceLevelObjective>
46     <wsag:BusinessValueList>
47      <wsag:CustomBusinessValue
48      xmlns:strat="http://www.telecom-sudparis.com/strat">
49       <strategy:Ready_D>
50        Duplication condition
51       </strategy:Ready_D>
52       <strategy:Ready_C>
53        Consolidation condition
54       </strategy:Ready_C>
55       ...
56      </wsag:CustomBusinessValue>
57      <wsag:CustomBusinessValue
58      xmlns:actio="http://www.telecom-sudparis.com/actio">
59       <strat:ReconfigurationAction>
60       Duplication
61       </strat:ReconfigurationAction>
62       <strat:ReconfigurationAction>
```

```
63        Consolidation
64       </strat:ReconfigurationAction>
65        ...
66      </wsag:CustomBusinessValue>
67     </wsag:BusinessValueList>
68    </wsag:GuaranteeTerm>
69   </wsag:All>
70  </wsag:Terms>
71 </wsag:AgreementOffer>
```

### 5.4.3 Experiment

In order to validate our work, we implemented an OCCI server that supports the previously described resources. We also implemented the OCCI compliant API that allows to seamlessly interacting with different PaaS Providers. Then, we realized preliminary experiments to test the efficiency of our proposal. In the following, we describe the different aspects of our implementation as well as the experiment results.

#### 5.4.3.1 Use Case

To experiment our work we propose a real use case that consists on the deployment of a SBP on the public PaaS CloudFoundry [85]. We would like to establish an autonomic infrastructure to render the SBP elastic.

For the SaaS and PaaS descriptions we are using a generic model that was proposed in [86]. We use also a generic API called COAPS [87][88] that allows to seamlessly interact with different PaaS in a generic manner. This API exposes a set of generic and RESTful HTTP operations (i.e. GET, POST, PUT and DELETE) for Cloud applications management and provisioning regarding the target PaaS. It provides an abstraction layer for existing PaaS allowing PaaS application provisioning in a unified manner.

To render the SBP elastic, we chose to use thresholds-based elasticity strategies that base their duplication/consolidation decisions on SBP response time metric. Consequently, we need to gather monitoring data related to the response time of each service and analyze these data. If the response time is over the max threshold we need to trigger a duplication action traduced to a call to COAPS to add more instances of the application. In the other case, if the response time is under the min threshold, we need to trigger a consolidation action traduced to a call to COAPS to remove an instance of the application. The different thresholds, analysis strategies and reconfiguration actions could be extracted from a previously defined SLA contract (Listing 5.1).

The input of our OCCI Server is a SLA describing the SBP and its requirements as well as the needed details for the autonomic aspects. It is passed to the *Autonomic Manager* in the following HTTP POST request:

```
> POST http://localhost:8182/amanager
> Category: amanager; class="kind";
X-OCCI-Attribute: name=AManager
X-OCCI-Attribute: version=1
X-OCCI-Attribute: slalocation=location
Content-Type: text/occi
```

The *Autonomic Manager* uses the SLA to build the environment where to deploy the SBP. It carries on the list of queries to the OCCI Server to deploy and start the different services after adding the needed Mixins (i.e., monitoring and reconfiguration Mixins). To deploy and start the services, the OCCI Server uses the actions proposed by COAPS to this end. In the realized use case, the Subscription Mixin uses a *NewRelic* monitoring service [83] to get monitoring data.

After the provisioning of the SBP, the *Autonomic Manager* proceeds to the instantiation of all the needed Resources and Links via the OCCI Server. From the SLA, the *Autonomic Manager* detects the need to monitor the response time of each service. It extracts also the defined thresholds as well as the analysis elasticity strategies and the reconfiguration actions. These information are described using user specific elements that we added to WS-Agreement. As shown in Listing 5.1 example the max and min thresholds of the service $s2$ are fixed respectively to 1000 and 500 ms. The monitoring data gathered by the Subscription Mixin are parsed by this latter. It extracts the response time of each service and sends eventual notifications to the *Analyzer*. For each notification, the *Analyzer* applies its elasticity strategies to verify that the response time is between min and max thresholds. If the SLA is not respected, the *Analyzer* resource triggers an alert targeting the *Planner*. This *Planner* generates reconfiguration actions to duplicate or consolidate the concerned service.

At this step, we showed how the *Autonomic Manager* can instantiate the autonomic loop with all the needed Resources, Links and Mixins. A usage example of this infrastructure is when a service of the SBP is overloaded. The response time of this service changes and becomes greater than max threshold. Using its *Subscription* Mixin, the concerned service generates a notification containing the new value of the monitored attribute. At the reception of the notification, this latter applies its strategies to verify if the SLA is violated or not. If the $ready_D$ is verified. The *Analyzer* raises an alert and send it to the *Planner*. Receiving this alert, the *Planner* generates the needed reconfiguration actions to ensure the *Duplication*. This latter asks COAPS to add more instances to the concerned service.

In [89], a description of the realized work is available. The page contains a link to download an archive of all the implementations. The archive includes also a user guide document that explains how to test the project using a SBP. In the same page, the client that we used to invoke the SBP is available.

#### 5.4.3.2 Evaluation

In order to validate our proposal, we present in this section the use case that we realized to this end. We describe the experiments environment and the preliminary results.

**Evaluation Environment**

To perform our experiments we used the NCF (Network and Cloud Federation) experimental platform deployed in our laboratory and described in Section 5.3.3.2.

In our tests we were focused on the PaaS and SaaS layers. Specifically, we deploy a SBP (i.e, SaaS) on the public PaaS CloudFoundry. Our goal is to show how we specialize an autonomic infrastructure to ensure SBPs elasticity.

For this aim, we implement the SBP of the online computer shopping service presented previously. We included the needed information related to the deployment of the application and the establishment of the infrastructure in a SLA expressed with the extended WS-Agreement that we presented in Section 5.4.3.1. Afterwards, we run our OCCI Server on VMs deployed by OpenNebula IaaS manager. Then, we instantiate an *Autonomic Manager* to deploy and start the different services of the SBP enforced with monitoring and reconfiguration Mixins on CloudFoundry using COAPS. After that, it continues the establishment of the previously defined Resources in order to build the autonomic infrastructure.

**Evaluation Result**

To show the efficiency of a solution there are different criteria to take into account. For our experiments, we fixed three criteria to verify:

- The time of the establishment of our infrastructure must be acceptable compared with the time of the Resource (the SBP in this use case) deployment;
- The time of the reconfiguration of the Resource (i.e., the SBP) must be convincing compared with the deployment time of a new Resource;
- The Autonomic infrastructure must enhance the QoS of the managed Resource (i.e., in our use case it must improve the SBP response time).

For the first criteria, we took different measurements of the time needed to deploy and start a SBP and the time needed to build the related autonomic infrastructure. The measurements show that the deployment and starting time depends on the size of the services of the SBP. However, the establishment of the rest of the autonomic infrastructure is independent of the Resource. To deploy our test SBP composed of four services to CloudFoundry we measured 152.09 seconds. To start this same SBP we measured 33.9 seconds. The time needed to establish the autonomic infrastructure

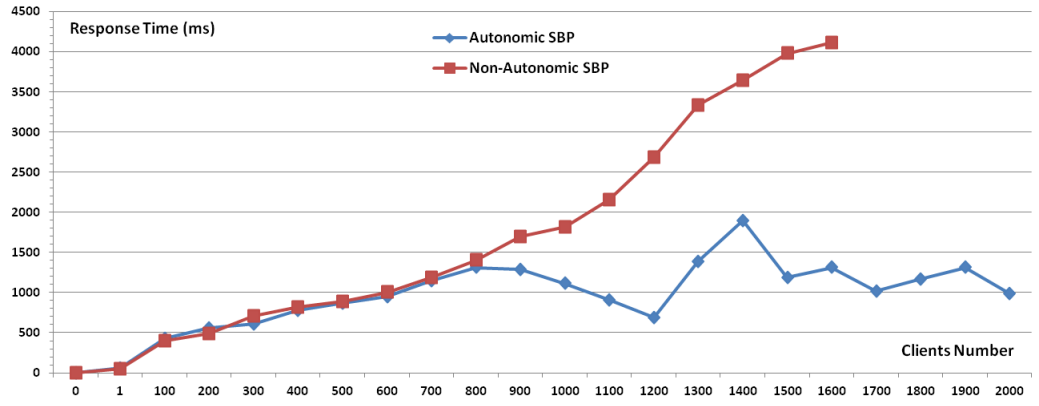


Figure 5.14: Response time of the SBP before and after adding our Autonomic Infrastructure.

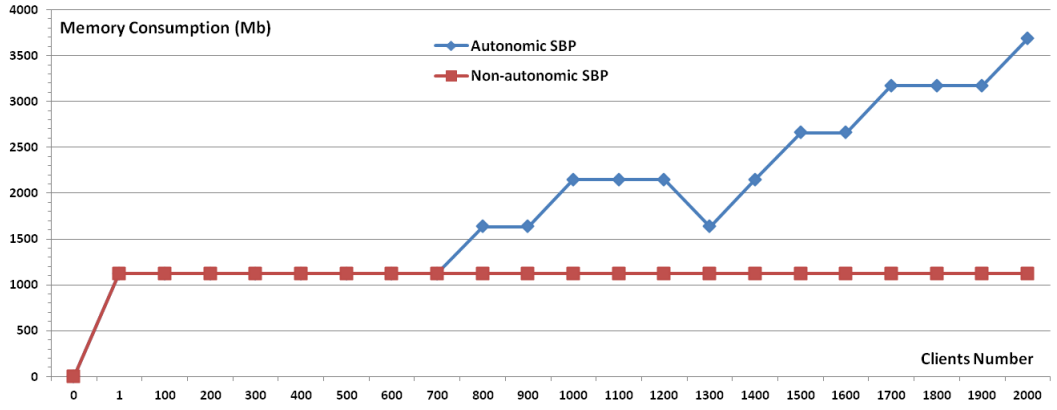


Figure 5.15: Memory consumption of the SBP before and after adding our Autonomic Infrastructure.

is independent of the SBP size and is around 15.9 seconds. This time is encouraging since it is negligible compared to the SBP deployment and starting time.

During this series of experiments, we measured also the needed time to apply a reconfiguration action (i.e., duplication or consolidation). The needed time to reconfigure a service is about 11.6 seconds. Indeed, this time is negligible compared to the needed time to redeploy and start the SBP.

In order to validate the third criteria that we fixed, we had to compare the QoS (i.e., response time in our case) of the SBP in CloudFoundry before and after adding our autonomic infrastructure. To this end, we developed a REST Client able to call

the deployed SBP. We deployed this client on different VMs in order to launch a big number of parallel calls. This is to create a client query burst targeting the SBP. In each series of these experiments we modified the number of the used clients targeting our SBP and we saved the measured response times. The results of this evaluation are shown in Figure 5.14.

The response time of the SBP without our mechanisms is increasing proportionally with the number of clients queries. Moreover, after 1600 clients, the SBP went down. In contrast, using our mechanisms, whenever the response time reached the max threshold that we specified in the SLA (i.e., 1000 ms for the service $s2$ in this case), a reconfiguration action is triggered to duplicate the overloaded service by adding a new instance to it. This reconfiguration decreases the response time of all the SBP. If the response time is under the min threshold that we specified in the SLA(i.e., 500 ms for the service $s2$ in this case), a reconfiguration action is triggered to consolidate the service instances. The experiments show the efficiency of our approach to enhance the behavior of the SBP. In fact, over 700 clients, the response time remains around the specified thresholds. The experiments show also that we reached 2000 clients without any downtime of the SBP. In some points, the response time can be over the max threshold, this is due to the number of clients queries sent before the elasticity action takes effect.

Over and above the response time of the SBP, we measured the memory consumption of our SBP. The results in Figure 5.15 shows the difference between the memory consumption of a traditional SBP and an elastic one. The memory consumption of a traditional SBP is constant since each service has its allocated memory and no new instances are added. Meanwhile, for an elastic SBP, the curve shows that the memory consumption is changing whenever there is a duplication or a consolidation. The cost of a duplication correspond to 512 mb that represents the size of an added service instance.

In this section, we proposed an autonomic approach for managing SBPs elasticity on Cloud platforms. To this end, we propose an approach for integrating our elasticity controller in an autonomic infrastructure in order to provide SBPs with elasticity facilities. In addition, we proved that our approach is efficient when applied on a realistic situation. The obtained results show that we verify the different considered criteria.

## 5.5 Conclusion

In this chapter, we proposed two approaches for the provisioning of SBPs elasticity in the Cloud. On one hand, we proposed an end-to-end approach for the provisioning of elastic SBPs on Cloud infrastructures by packaging SBPs in elastic micro-containers. On the other hand, we proposed an autonomic approach to manage SBPs elasticity on Cloud platforms by integrating our elasticity controller in an autonomic infrastructure for Cloud resources. We showed the feasibility and efficiency of our elasticity approach

in addressing elasticity in the Cloud. We presented the different experiments that we realized to validate and prove the efficiency of our work in real Cloud environments. Indeed, the results are encouraging. At the same time, they opened new perspectives.

Chapter 6

# Conclusion

**Contents**


**6.1 Contributions** . . . . . **125**

**6.2 Perspectives** . . . . . **126**

6.2.1 Short-term perspectives . . . . . 126

6.2.2 Long-term perspectives: Mixing Multi-tenancy and Elasticity of Business Processes . . . . . 127


## 6.1 Contributions

In this thesis, we dressed a list of objectives in order to provide a solution that satisfies our requirements for modeling, evaluating and provisioning of SBPs elasticity in the Cloud. We started by studying different existing works related to elasticity. Our study of the state of the art was divided into three parts treating respectively models and mechanisms of elasticity, evaluation of elasticity and provisioning of elasticity mechanisms in the Cloud. Afterwards, we made a synthesis of the presented works to recapitulate the advantages and drawbacks of each one. Consequently, we proposed three major contributions in order to cover the different objectives that we dressed for our solution.

As a first contribution, we proposed a formal model for SBPs elasticity that intertwines duplication/consolidation operations with an elasticity controller. To do this, we modeled SBPs using Petri nets and formalized elasticity operations (duplication/-consolidation) that operate at the scope of services. We showed that our model can ensure the elasticity of stateless, stateful and timed SBPS while preserving the semantics of these latter. In addition, we proposed a generic controller, modeled using high level Petri nets, that monitors SBP execution, analyzes monitoring information (based on elasticity strategies) and executes elasticity actions (duplication/consolidation) in order to ensure SBPs elasticity.

After facing the challenge of defining a formal model for ensuring SBPs elasticity at the scope of services, we were interested in the evaluation of elasticity strategies that are used by the elasticity controller to decide on when, where and how to use duplication/consolidation operations. To this end, we proposed an approach that uses the

previously defined generic controller as a framework for the validation and evaluation of different elasticity strategies before using them in real Cloud environments. Our evaluation approach allows validating the correctness of elasticity strategies (using verification) and comparing between strategies (using simulation).

In order to go further in our reasoning, we proposed to provide mechanisms for the provisioning of elastic SBPs in the Cloud. In fact, any proposed approach for ensuring SBPs elasticity needs to be validated in real Cloud environments. To this end, we proposed two approaches for the provisioning of elastic SBPs while considering two provisioning contexts: IaaS and PaaS. The first approach consists in an end-to-end approach for the provisioning of elastic SBPs on Cloud infrastructures. This approach packages non-elastic SBPs in micro-containers, extended with our elasticity mechanisms, before deploying them in real Cloud infrastructures. The second approach consists in an autonomic approach for managing SBPs elasticity on Cloud platforms. This approach integrates our elasticity controller in an autonomic infrastructure in order to dynamically add elasticity facilities to SBPs in real Cloud platforms.

The contributions that we presented in this manuscript respect the research objectives that we dressed in the beginning of this work.

## 6.2 Perspectives

In this thesis, we face different complex problems regarding SBPs elasticity. We solved several research problems and we included others in our future work. We divided our future work to short-term and long-term perspectives.

### 6.2.1 Short-term perspectives

As short-term perspectives, we aim at addressing different research problems by extending our already done work with three extensions:

Currently, our elasticity approach considers the workload of services (in terms of number of calls) as metrics to take elasticity decisions. In addition, all service calls are treated in the same manner. As a first extension of our work, we will consider additional functional and non-functional properties. To this end, we should identify such properties, define how to monitor them and develop mechanisms to configure elasticity according to them. For example, we can consider a price property that allows ensuring SBPs elasticity while respecting a given budget fixed by the user (i.e., we can use the price property as an elasticity constraint). Another example consists in refining our SBP model to consider different kinds of service calls (i.e., be able to differentiate between calls). In fact, it would be interesting to be able to identify different groups of calls ranging from the most priority ones to the ones that can suffer from loss of QoS. To do this, we can annotate service calls (tokens) with colors (i.e., each color will have specific characteristics).

The second extension consists in providing a Domain Specific Language (DSL) for defining reactive, predictive and hybrid elasticity strategies. It would be interesting to have a language for defining elasticity strategies in a unified manner while considering different kinds of strategies.

The last extension consists in investigating different scenarios for the deployment of elasticity controllers in the Cloud. In fact, a controller can be assigned for all the deployed SBPs, for each subset of SBPs or even for each deployed SBP. To this end, we plan to consider the collaboration between deployed controllers in order to achieve a global elasticity objective.

### 6.2.2 Long-term perspectives: Mixing Multi-tenancy and Elasticity of Business Processes

Cloud computing is particularly interesting in situations where many organizations need to support similar processes. For example, smart cities, administrations, agencies, etc. all need to support similar processes but they also need local and controlled variations of these similar processes. Therefore, Cloud platforms should provision multi-tenancy mechanisms such that business processes can be customized to tenants while allowing them to share resources. In fact, it is not realistic to enforce "one size fits all" as tenants may have different needs and preferences.

There are mainly three approaches developed for multi-tenancy in Cloud environments [90]. These approaches require varying the degree of resource sharing and the development complexity:

- Approach 1. sharing a single business process instance among all tenants,
- Approach 2. running tenant specific process instances on a shared process engine,
- Approach 3. running tenant specific process instances on a dedicated process engine.

Approaches 2 and 3 are based on deployment techniques. They provide a very low level degree of resource sharing. In approach 1, all tenants share the OS, the process engine and a single instance of the business process. This is accomplished by parameterizing a single instance of a business process with a tenant identification parameter. However, the drawback of this approach is its high development complexity which could prevent democratization of such technique.

Our objective is to extend our approach for business processes elasticity with multi-tenancy mechanisms in order to ensure different preferences of tenants (holders). To do this, we aim at using configurable business processes to support multi-tenancy and elasticity mechanisms in an integrated way. Actually, we propose to use configurable languages not only for modeling purpose, as it is the case in the state of the art, but also for supporting multi-tenancy and elasticity at the runtime. To this

end, we will (1) define an operational semantics for a particular configurable language (C-WF-nets) to support multi-tenancy, and (2) extend control mechanisms to decide about how, when and where elasticity is applied while taking into account the multi-tenancy property (i.e., Identifying parts or services that are elastic, multi-tenant, elastic and multi-tenant or mono-tenant and non elastic).

# Bibliography


[1] NIST, "Final Version of NIST Cloud Computing Definition Published." http://www.nist.gov/itl/csd/cloud-102511.cfm, 2011.

[2] W.-T. Tsai, X. Sun, Q. Shao, and G. Qi, "Two-tier multi-tenancy scaling and load balancing," in *IEEE International Conference on E-Business Engineering*, 2010, pp. 484–489.

[3] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for bpel workflows using amazon's elastic compute cloud," in *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2009, pp. 140–147.

[4] J. Loff and J. Garcia, "Vadara: Predictive elasticity for cloud applications," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2014, pp. 541–546.

[5] R.-H. Hwang, C.-N. Lee, Y.-R. Chen, and D.-J. Zhang-Jian, "Cost optimization of elasticity cloud resource subscription policy," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[6] B. Suleiman and S. Venugopal, "Modeling performance of elasticity rules for cloud-based applications," in *17th IEEE International Conference on Enterprise Distributed Object Computing (EDOC).*, Sept 2013, pp. 201–206.

[7] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Cloud elasticity using probabilistic model checking," *CoRR*, vol. abs/1405.4699, 2014. [Online]. Available: http://arxiv.org/abs/1405.4699

[8] G. Copil, D. Trihinas, H.-L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. Dikaiakos, "Advise – a framework for evaluating cloud service elasticity behavior," in *International Conference on Service Oriented Computing (ICSOC)*, ser. Lecture Notes in Computer Science, vol. 8831. Springer Berlin Heidelberg, 2014, pp. 275–290.

[9] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Introducing the vienna platform for elastic processes," in *International Conference on Service Oriented Computing (ICSOC) Workshops*, 2013, pp. 179–190.

[10] M. Dhingra, J. Lakshmi, S. Nandy, C. Bhattacharyya, and K. Gopinath, "Elastic resources framework in iaas, preserving performance slas," in *International Conference on Cloud Computing (IEEE CLOUD)*, June 2013, pp. 430–437.

[11] 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 *The Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2012, pp. 1042–1049.

[12] P. D. Kaur and I. Chana, "A resource elasticity framework for qos-aware execution of cloud applications," *Future Generation Computer Systems*, vol. 37, no. 0, pp. 14 – 25, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X14000430

[13] L. Letondeur, X. Etchevers, T. Coupaye, F. Boyer, and N. de Palma, "Architectural Model and Planification Algorithm for the Self-Management of Elastic Cloud Applications." in *IEEE International Conference on Cloud and Autonomic Computing*, 2014.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[15] L. Wu, S. Garg, S. Versteeg, and R. Buyya, "Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments," *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 465–485, July 2014.

[16] Cloud Industry Forum, "UK Cloud adoption and trends for 2013," Sword House, Totteridge Road, High Wycombe HP13 6DG, Tech. Rep., 2012. [Online]. Available: http://www.cloudindustryforum.org/white-papers/uk-cloud-adoption-and-trends-for-2013

[17] M. Papazoglou and W. van den Heuvel, "Blueprinting the cloud," *Internet Computing, IEEE*, vol. 15, no. 6, pp. 74–79, Nov 2011.

[18] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong, "Principles of elastic processes," *IEEE Internet Computing*, vol. 15, no. 5, pp. 66–71, 2011.

[19] 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," in *http://virtualization.sys-con.com/node/612375*, 2009. [Online]. Available: http://virtualization.sys-con.com/node/612375

[20] 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*, ser. NCM. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–51.

[21] G. Galante and L. de Bona, "A survey on cloud computing elasticity," in *IEEE International Conference on Utility and Cloud Computing (UCC)*, 2012, pp. 263–270.

[22] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic business process management: State of the art and open challenges for bpm in the cloud," *Future Generation Computer Systems*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1400168X

[23] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture," *W3C Working Group Note*, vol. 11, pp. 2005–1, 2004.

[24] S. Burbeck, "The Tao of e-business services: The evolution of Web applications into service-oriented components with Web services," IBM Software Group, 2000.

[25] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Dec. 2003, pp. 3–12.

[26] L. Liu and M. T. Özsu, Eds., *Encyclopedia of Database Systems*. Springer US, 2009.

[27] D. Shawky and A. Ali, "Defining a measure of cloud computing elasticity," in *2012 1st International Conference on Systems and Computer Science (ICSCS)*, Aug 2012, pp. 1–5.

[28] K. Klai and S. Tata, "Formal modeling of elastic service-based business processes," in *International Conference on Services Computing (IEEE SCC)*, 2013, pp. 424–431.

[29] C. Chrysoulas, G. Kostopoulos, E. Haleplidis, R. Haas, S. Denazis, and O. Koufopavlou, "A decision making framework for dynamic service deployment," in *IST Mobile and Wireless Communications Summit*, 2006.

[30] J. B. Weissman, S. Kim, and D. England, "A framework for dynamic service adaptation in the grid: Next generation software program progress report," in *International Parallel and Distributed Processing Symposium*. Los Alamitos, CA, USA: IEEE Computer Society, 2005.

[31] W. Zhao, P. Melliar-Smith, and L. Moser, "Fault tolerance middleware for cloud computing," in *International Conference on Cloud Computing (IEEE CLOUD)*, 2010, pp. 67–74.

[32] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, no. 0, pp. 82 – 98, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001148

[33] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium (NOMS)*, April 2012, pp. 204–212.

[34] D. Perez-Palacin, R. Mirandola, and J. Merseguer, "Qos and energy management with petri nets: A self-adaptive framework," *Journal of Systems and Software*, vol. 85, pp. 2796 – 2811, 2012.

[35] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2–19, 2012.

[36] Y. Song, Y. Sun, and W. Shi, "A two-tiered on-demand resource allocation mechanism for vm-based data centers," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 116–129, 2013.

[37] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 85–96.

[38] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 23–27.

[39] ——, "Elasticity in Cloud Computing: What it is, and What it is Not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*. USENIX, June 2013.

[40] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal, "Self-adaptive resource allocation for elastic process execution," in *International Conference on Cloud Computing (IEEE CLOUD)*, 2013, pp. 220–227.

[41] P. Hoenisch, S. Schulte, and S. Dustdar, "Workflow scheduling and resource allocation for cloud-based execution of elastic processes," in *International Conference on Service-Oriented Computing and Applications (SOCA)*, 2013, pp. 1–8.

[42] L.-S. Lê, H. L. Truong, A. Ghose, and S. Dustdar, "On elasticity and constrainedness of business services provisioning," in *International Conference on Services Computing (IEEE SCC)*, 2012, pp. 384–391.

[43] L. Letondeur, "Planification for autonomic management of elasticity for applications in the cloud," Theses, Universite Joseph Fourier, Oct. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/tel-01140128

[44] M. Amziani, T. Melliti, and S. Tata, "A generic framework for service-based business process elasticity in the cloud," in *International Conference on Business Process Management (BPM)*, 2012, pp. 194–199.

[45] ——, "Formal modeling and evaluation of stateful service-based business process elasticity in the cloud," in *International Conference on Cooperative Information Systems (CoopIS)*, 2013.

[46] M. Amziani, K. Klai, T. Melliti, and S. Tata, "Time-based evaluation of service-based business process elasticity in the cloud," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013.

[47] M. Amziani, T. Melliti, and S. Tata, "Formal modeling and evaluation of service-based business process elasticity in the cloud," in *IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2013.

[48] Stephen A. White, "Introduction to BPMN," Tech. Rep., 2006. [Online]. Available: http://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf

[49] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*.

[50] W. M. P. VAN DER AALST, "The application of petri nets to workflow management," *Journal of Circuits, Systems and Computers*, vol. 08, no. 01, pp. 21–66, 1998.

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

[52] L. Jacobsen, M. Jacobsen, M. H. Moller, and J. Srba, "Verification of timed-arc petri nets," *SOFSEM 2011: Theory and Practice of Computer Science*, vol. 6543, pp. 46–72, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18381-2_4

[53] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, *A Practical Guide to the IBM Autonomic Computing Toolkit*, 2004. [Online]. Available: http://books.google.fr/books?id=XHeoSgAACAAJ

[54] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *International Conference on Cloud Computing (IEEE CLOUD)*, 2011.

[55] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 45–52, Jan. 2011.

[56] J. Esparza and M. Nielsen, "Decidability issues for petri nets - a survey," 1994.

[57] L. M. Kristensen, *State Space Methods for Coloured Petri Nets.* Ph.D. Dissertation, 2000.

[58] H. M. W. E. Verbeek, A. J. Pretorius, W. M. P. van der Aalst, and J. J. van Wijk, "Assessing state spaces using petri-net synthesis and attribute-based visualization," *T. Petri Nets and Other Models of Concurrency*, vol. 1, pp. 152–171, 2008.

[59] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

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

[61] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *International Conference on Advanced Information Networking and Applications (AINA).* Los Alamitos, CA, USA: IEEE Computer Society, 2012.

[62] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *The IEEE International Conference on e-Business Engineering (ICEBE)*, 2009.

[63] S. Yangui, M. Mohamed, S. Tata, and S. Moalla, "Scalable Service Containers." in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 348–356.

[64] M. Mohamed, S. Yangui, S. Moalla, and S. Tata, "Web service micro-container for service-based applications in cloud environments," in *IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2011.

[65] M. Mohamed, D. Belaïd, and S. Tata, "How to Provide Monitoring Facilities to Services When They Are Deployed in the Cloud?" in *International Conference on Cloud Computing and Services Science (CLOSER)*, 2012.

[66] "WSDL," http://www.w3.org/TR/wsdl, 2014.

[67] A. Omezzine, S. Yangui, N. Bellamine, and S. Tata, "Mobile Service Microcontainers for Cloud Environments," in *IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, June 2012, pp. 154–160.

[68] "PNML," http://www.pnml.org/, 2014.

[69] OpenNebula, "OpenNebula: Flexible Entreprise Cloud Made simple," http://opennebula.org, 2014.

[70] OpenStack, "Openstack: Open source software for building private and public clouds." http://www.openstack.org, 2014.

[71] M. Mohamed, D. Belaïd, and S. Tata, "Monitoring and Reconfiguration for OCCI Resources," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013.

[72] M. Mohamed, M. Amziani, D. Belaïd, S. Tata, and T. Melliti, "An autonomic approach to manage elasticity of business processes in the cloud," *Future Generation Computer Systems*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X14002106

[73] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart, "An architectural approach to autonomic computing," in *Autonomic Computing, 2004. Proceedings. International Conference on*, May 2004, pp. 2–9.

[74] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, Jan. 2003. [Online]. Available: http://dx.doi.org/10.1147/sj.421.0005

[75] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface - Core," Tech. Rep., 2011. [Online]. Available: http://www.ogf.org/documents/GFD.183.pdf

[76] T. Metsch and A. Edmonds, "Open Cloud Computing Interface - RESTful HTTP Rendering," Tech. Rep., 2011. [Online]. Available: http://www.ogf.org/documents/GFD.185.pdf

[77] ——, "Open Cloud Computing Interface - Infrastructure," Tech. Rep., 2011. [Online]. Available: http://www.ogf.org/documents/GFD.184.pdf

[78] S. Yangui, M. Mohamed, M. Sellami, and S. Tata, "Open Cloud Computing Interface - Platform," Tech. Rep., 2013. [Online]. Available: http://www-inf.int-evry.fr/SIMBAD/tools/OCCI/occi-platform.pdf

[79] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "Open Cloud Computing Interface - Application," Tech. Rep., 2013. [Online]. Available: http://www-inf.int-evry.fr/SIMBAD/tools/OCCI/occi-application.pdf

[80] "Occi4Java," https://github.com/occi4java/occi4java, 2014.

[81] "Java-mixins," http://hg.berniecode.com/java-mixins, 2014.

[82] "Restlet," http://restlet.org/, 2014.

[83] Cloud Foundry Blog, "Monitoring Cloud Foundry Applications with New Relic," 2014. [Online]. Available: http://blog.cloudfoundry.com/2013/10/10/monitoring-cloud-foundry-applications-with-new-relic

[84] M. Mohamed, D. Belaïd, and S. Tata, "Adding Monitoring and Reconfiguration Facilities for Service-Based Applications in the Cloud," in *International Conference on Advanced Information Networking and Applications (AINA)*, 2013.

[85] "CloudFoundry," http://www.cloudfoundry.com/, 2014.

[86] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "PaaS-independent Provisioning and Management of Applications in the Cloud," in *International Conference on Cloud Computing (IEEE CLOUD)*, 2013.

[87] "COAPS: A Generic Cloud Application Provisioning and Management API," http://www-inf.it-sudparis.eu/SIMBAD/tools/COAPS/, 2014.

[88] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, "CompatibleOne: The Open Source Cloud Broker," *Journal of Grid Computing*, vol. 12, no. 1, 2014.

[89] "OCCI4Java and Autonomic Computing: A java implementation for the OCCI Autonomic Comuting extension," http://www-inf.it-sudparis.eu/SIMBAD/tools/OCCI/autonomic/, 2014.

[90] C. Osipov G. Goldszmidt M. Taylor and I. Poddar, "Develop and Deploy Multi-tenant Web-Delivered Solutions Using IBM Middleware, Part 1: Challenges and Architectural Patterns," Tech. Rep., 2009.