Description and evaluation of elasticity strategies for business processes in the Cloud

Aïcha Ben Jrad

NNT : 2019SACLL012

Description and Evaluation of Elasticity Strategies for Business Processes in the Cloud

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom SudParis et l'Ecole Nationale d'Ingénieurs de Tunis

École doctorale n°580 :
Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Tunis, le 05/07/2019, par

Aïcha Ben Jrad

Composition du Jury :

M. Nejib Ben Hadj- Alouane Professeur, ENIT, Université Tunis El Manar, Tunisie	Président
Mme. Parisa Ghodous Professeur, Univeristé de Lyon 1, France	Rapporteur
Mme. Sonia Ayachi Ghannouchi Maître de conférences (HDR), ISG de Sousse, Université de Sousse, Tunisie	Rapporteur
M. Jean-Marc Delosme Professeur, Université d'Evry-Val d'Essonne, France	Examinateur
M. Bruno Defude Professeur, Télécom SudParis, France	Examinateur
M. Samir Tata Professeur, Télécom SudParis (SAMOVAR), France	Directeur de thèse
M. Sami Bhiri Maître de conférences (HDR), ISIMM, Université de Monastir, (OASIS), Tunisie	Co-Directeur de thèse

Thèse de doctorat

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Description et évaluation de stratégies d'élasticité des processus métiers dans le Cloud

Mots clés : Applications à base de services, Stratégies d'élasticité, Evaluation, Modèle d'élasticité, Cloud Computing.

Résumé : De nos jours, de plus en plus d'entreprises migrent leurs applications à base de services (AbSs) vers le Cloud. L'élasticité du Cloud Computing est la caractéristique principale derrière ce phénomène de migration. Le principe d'élasticité a attiré beaucoup d'attention ces dernières années comme une tâche pivot qui permet d'assurer un bon compromis entre les Qualités de service désirées et les coûts opérationnels des AbSs. Toutefois, le contrôle d'élasticité des AbSs et la définition des stratégies d'élasticité non-triviales sont encore des tâches difficiles à réaliser. La difficulté de ces tâches est de plus accentuée avec l'absence d'un langage unifié pour exprimer ces stratégies et un moyen pour les évaluer et les valider avant de les utiliser dans un environnement Cloud réel. Dans notre travail, nous nous intéressons à la définition et l'évaluation des stratégies d'élasticité des applications à base de services dans le Cloud.

Dans ce contexte, nous avons proposé un framework d'évaluation des stratégies d'élasticité basé sur des mèthodes formelles pour evaluer le comportement des stratégies. Pour cela, nous avons défini un modèle formel pour l'élasticité des AbSs dans le cloud. Nous avons modélisé le modèle de déploiement des AbSs en utilisant les réseaux de petri et défini des opérations d'élasticité pour l'élasticité hybrid. Le modèle proposé permet de décrire les caractéristiques des services composant un AbS et leurs requêtes afin de définir des stratégies non-triviales. Après la modèlisation de l'élasticité des AbSs, deux langages dédiés, nommé STRATModel et STRAT, sont proposés pour faciliter la description des stratégies d'élasticité qui sont basées sur des modèles d'élasticité différents. STRATModel permet de définir des modèles d'élasticité et de générer leur contrôleurs associés qui seront utilisés pour contrôler l'élasticité des AbSs et évaluer les stratégies. En se basant sur le langage STRATModel, STRAT est proposé pour décrire d'une manière unifiée des stratégies d'élasticité pour des AbSs. Il est défini comme un langage à base des règles permettant de définir un ensemble des conditions pour des actions spécifiées dans un modèle STRATModel. En prennant ces langages et notre modèle formel comme une base, un framework, nommé STRATFram, est proposé pour être utilisé par les responsables de configuration des AbSs comme un support qui leur permet de décrire et d'évaluer leur stratégies d'élasticité avant de les utiliser dans le cloud. L'évaluation des stratégies consiste à fournir un ensemble de courbes permettant l'analyse et la comparaison de leur comportement. Nos contributions et développements permettent aux responsable des AbSs de choisir les stratégies d'élasticité les plus adaptées à leurs AbSs.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title : Description and evaluation of elasticity strategies for business processes in the cloud

Keywords : Service-based business process, Elasticity Strategies, Evaluation, Elasticity Model, Cloud Computing.

Abstract : In the recent years, growing attention has been paid to the concept of Cloud Computing as a new computing paradigm for executing and handling business processes in an efficient and cost-effective way. Cloud Computing's elasticity and its flexibility in service delivery are the most important features behind this attention which encourage companies to migrate their operation/processes to the cloud to ensure the required Quality of Service (QoS) while using resources and reduce their expenses. Elasticity management has been considered as a pivotal issue among IT community that works on finding the right tradeoffs between QoS levels and operational costs by developing novel methods and mechanisms. However, controlling process elasticity and defining non-trivial elasticity strategies are still challenging tasks. Also, despite the growing attention paid to the cloud and its elasticity property in particular, there is still a lack of solutions that support the evaluation of elasticity strategies used to ensure the elasticity of business processes at service-level before using them in real cloud environments.

In this thesis, a framework for describing and evaluating elasticity strategies for service-based business processes (SBPs) is proposed. Starting from representing elastic SBPs, a formal elasticity approach is proposed to allow the evaluation of elasticity strategies before using them in the cloud. The proposed approach is composed of (i) a formal model, that describes elastic execution environments for SBPs while considering the distinguish requirements for services requests, and (ii) elasticity operations/capabilities for hybrid scaling that are defined and formalized to illustrate their application on the model to ensure its elasticity. The proposed model is defined based on high-level petri nets to describe not only the characteristics of service engines, hosting SBPs services, but also the characteristics of their requests that allows defining more sophisticated elasticity strategies. After modeling SBPs elasticity for the purpose of evaluating elasticity strategies, two domain-specific languages, named STRATModel and STRAT, are designed to be used together to describe elasticity strategies for different elasticity models. STRATModel language allows describing elasticity models with different elasticity capabilities and generating their associated elasticity controllers that are used to manage SBPs elasticity and evaluate strategies. Based on STRATModel language, STRAT language is designed to allow specifying elasticity strategies governing SBP elasticity according to a given elasticity model written using STRATModel language. With the formal model and the designed languages as basis, a framework, named STRATFram, is proposed in order to provide SBPs holders a support on which they can describe and evaluate their elasticity strategies before investing in using them in the cloud. The evaluation consists in providing a set of plots that allows the analysis and the comparison of strategies. Our contributions and developments provide SBP holders with facilities to choose elasticity strategies that fit to their elastic SBPs and usage behaviors.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

To my family, my parents,
for their unconditional love and support

Acknowledgements

It is with great pleasure that I reserve this page as a sign of profound gratitude to all the supportive people in my life who have given me strength, encouragement and support that enabled me to survive difficult times during my PhD journey. This thesis becomes a reality with their generous help and support.

I would like to start by thanking all the members of the jury for accepting to evaluate my work. My special appreciation goes to the committee president Professor Nejib Ben Hadj-Alouane *for accepting to chair my Phd defense, to the reading committee members Professor Parisa* Ghodous *and Dr. Sonia* Ayachi Ghannouchi *for their interest and to Professor Jean-Marc* Delosme *for being my thesis examiners.*

I owe my wholehearted gratitude to my supervisors Professor Samir Tata *and Dr. Sami* Bhiri *for their patient guidance and continuous support without which this thesis would not have been possible. Despite their many responsibilities, they have always found time to provide feedbacks, new ideas and suggestions on my work. I would like to thank them for allowing me to develop a better understanding of research and to grow as a research scientist. I would also like to express my appreciation to Professor Bruno* Defude *for accepting to be my co-thesis advisor and for his valuable advice, guidance and encouragement.*

I would also like to express my appreciation to Professor Bruno Defude *for his kindness and his valuable advice and for accepting to be in my Phd defense.*

I also want to give special thanks to all the members of the computer science department of Telecom SudParis, especially Brigitte Houassine *for her kind help and assistance. Many thanks to all my colleagues and friends inside Telecom SudParis and outside also for the excellent and truly enjoyable ambiance. My warmest thanks go to: Leila, Hayet, Nabila, Amina, Ines, Rania, Wided, Nouha, Marwa, Kaouther, Imen, Sarra, Mehdi and Tuanir. I am grateful to all of you, for your kindness, encouragements and support, and for the lovely moments we have spent together.*

Finally, I want to thank and dedicate this thesis to my family. I am eternally grateful to my parents, Mohamed and Selma, who have given me the chance of a good education, and so much love and support over the years. I probably owe them much more than they think. Thank you for having faith in me even when doubting myself. My warm thanks to my grand-father, Mohamed, and my sisters, Asma, Amina and Rabeb, for their love, their kindness and their continuous support. Last but not least, I am grateful to other members of my family and friends who have been there for me through ups and downs.

. . . Thank you all once again!

List of Publications

[1] "STRATFram: A framework for describing and evaluating Elasticity Strategies for Service-based business processes in the Cloud", Aicha Ben Jrad, Sami Bhiri and Samir Tata, Future Generation Computer Systems, vol. 97, pages 69-89, 2019. *(Journal Q1, Impact-factor = 4.639)*

[2] "STRATModel: Elasticity Model Description Language for evaluating Elasticity Strategies for Business Processes", Aicha Ben Jrad, Sami Bhiri and Samir Tata, in: On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE, Rhodes, Greece, October 23-27, 2017. *(Rank A)*

[3] "Data-Aware Modeling Of Elastic Processes For Elasticity Strategies Evaluation", Aicha Ben Jrad, Sami Bhiri and Samir Tata, in: IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, June 25-30, 2017. *(Rank B)*

[4] "Description and Evaluation of Elasticity Strategies for Business Processes in the Cloud", Aicha Ben Jrad, Sami Bhiri and Samir Tata, in: IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, June 27 - July 2, 2016. *(Rank A)*

Table of contents

List of Figures

List of Grammars

List of Listings

List of Tables

CHAPTER 1

Introduction

"Science is like a tree: contrary to popular belief,
both trees and science grow at their edges,
not at their core."
Peter Mika

Contents

1.1 General Context: *Elastic Service-based Business Processes*

Cloud Computing is gaining more and more importance in the Information Technologies (IT) scope as an emerging computing paradigm for managing and delivering services over the internet. One of the major assets of this paradigm is its economic model based on pay-as-you-go model. It allows the delivery of computing applications as a service rather than a product by enabling ubiquitous, convenient, and on-demand network access to large pools of computing resources (*e.g.*, computing, applications, services, storage, and network) that can be dynamically provisioned by increasing and decreasing services capacity to match workloads demands and usage optimization [Hogan 2011].

In today's information society, many companies of all sizes have started to make Cloud technology as the default technology that is used to handle their day-to-day activities, and more will do so increasingly in the near future. According to a survey

conducted by IDG Communications across different IT decision-makers at a variety of industries in 2018 [IDG 2018], nine out of ten companies already have at least some of their applications or a portion of computing infrastructure in the cloud or plan to move them to the cloud in the next 12 months.

The movement to the Cloud Computing as an IT infrastructure enables companies to operate more efficiently on the continuous incremental change in business operations/processes. By using it for executing their business processes, particularly their service-based business processes (SBPs) which consist of a set of elementary IT-enabled services, companies are able to remove the worry associated with the constraints and service interruptions that might occur in an internal data center infrastructure during peak periods. Especially in this digital age, many industry domains and business areas, such as eHealth [Mans 2010], manufacturing [Schulte 2014], or SmartGrids in the energy industry [Rohjans 2012], are dealing with a large number of SBP requests/instances, which might occur in a regular as well as in an ad-hoc manner, and/or the need to process a massive amount of data in particular tasks/services in a short time under specific Quality of Service (QoS) requirements which require a large amount of computational resources that can not be predicted in advance. The *rapid elasticity* feature of cloud computing has been the answer to such situations and one of the key drivers for companies to adopt cloud computing's technologies and move their business processes to it.

Elasticity is defined as the ability of a system to be adjustable to workload changes by provisioning as many resources as needed in autonomic manner in order to meet the required QoS [Herbst 2013]. Provisioning of resources can be made using horizontal elasticity, vertical elasticity, or the combination of both (*i.e.*, hybrid elasticity). Horizontal elasticity consists in replicating/removing instances of system elements to balance the current workload. It is also known as replication of resources. On the other hand, vertical elasticity consists in changing the characteristics/properties (*e.g.*, memory, CPU cores) of the used instances in the system by increasing or decreasing them. It is also known as resizing of resources. As a combination of horizontal and vertical elasticity, hybrid elasticity allows to add/remove instances with different characteristics. These elasticity capabilities are the main constructions of an elasticity model which defines the ground terms and functionalities that describe the elasticity of the managed system such as (1) the elasticity actions to be undertaken, (2) metrics to monitor to trigger the elasticity actions and (3) properties to access and reconfigure.

Elastic systems are usually composed of a managed system and a control system which is responsible on managing the former and ensuring its elasticity as shown in Figure 1.1. A control system mainly comprises an elasticity controller (EC) representing its functional component and elasticity information representing the basis for designing the controller and for its elasticity decisions. Elasticity information is categorized into: (*i*) an elasticity model (EM) which specifies the basic information (elasticity actions, metrics, *etc.*) on which the design of the EC is based, and (*ii*) an elasticity strategy (ES) which defines the rules for deciding when, where and how to use elasticity actions

(*e.g.*, adding or removing resources) defined in the EM. In other words, *the elasticity of the managed system is ensured by an EC, implementing a specific EM, according to an ES which is, in turn, defined based on the same EM.*

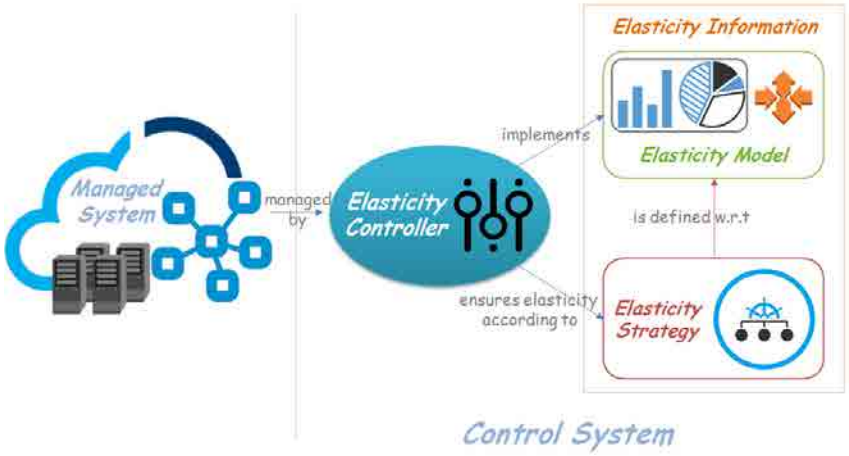


Figure 1.1: Elastic system architecture

The realisation of elastic systems has been the focus of several efforts in the recent years from IT community that worked on elasticity management to find the right tradeoffs between QoS levels and operational costs. Among these efforts, a new research trend in elastic systems has been emerged named *elastic processes* (or elastic SBPs) [Dustdar 2011] which refers to process automation and workflows that utilize cloud resources dynamically by acquiring and/or releasing them per workload changes to perform their tasks in an efficient and cost-effective way according to a predefined ES. The realisation of elastic processes requires taking into consideration the elasticity property of Cloud computing during business process life-cycle phases [Weske 2012]: including the definition of a SBP [Amziani 2015] and its enactment [Bessai 2012b, Bessai 2012a, Rosinosky 2016, Hoenisch 2014]. Notably, within this thesis, only the definition of SBPs is of further interest. The definition of a SBP is divided into two parts: (*i*) a conceptual definition and (*ii*) a technical definition. These parts correspond respectively to the first two phases in the business process life-cycle, namely (*i*) modeling and analysis and (*ii*) configuration. During the modeling and analysis phase, SBPs are identified, validated, and represented by SBP models which are used as a communication basis for SBP holders and help to refine and improve processes. Thereafter, in the configuration phase, technical information is provided to enhance the SBP model. This kind of information is used to facilitate its enactment by the business process management system (BPMS). In the cloud computing context, SBP holders are required to specify not only the cloud resources in which the SBP's services will be deployed but also any information/component that is needed to manage the process

elasticity in the cloud such as specifying the ES that should be used to control the SBP's elasticity and preserve the required QoS.

1.2 Problem Statement

Elastic systems in general and elastic SBPs in particular are still in their infancy, and their development and management face a lot of challenges that need to be tackled [Copil 2016]. One of the main challenges is specifying effective elasticity strategies that define requirements on the elasticity behavior of the system and able to prevent QoS violations. Specifying an elasticity strategy has been considered as a very hard task and likely to be doomed to fail if not defined with considerable care [Dutreilh 2010].

Usually, when developing elastic systems, elasticity strategies are hard-coded in the controller which makes defining complex strategies especially for SBPs using diverse metrics and information becomes easily cumbersome, error-prone and hard to maintain and to reuse. Moreover, many strategies can be defined for the same SBP where some of them are more effective than others.

Without providing a proper definition according to elastic SBP workload characteristics, elasticity strategies might lead to loss of QoS and large waste of resources. So, it is necessary to have a deeper understanding on how elasticity strategies behave and a high degree of expertise on how they should be specified to prevent (or minimize the occurrence of) unwanted results when using them in the cloud. However, this level of knowledge and expertise is hard to achieve in practice especially when we deal with a heterogeneous environment such the cloud and a large number of possible elasticity strategies that can be defined for a SBP. Furthermore, even with the need to find an effective elasticity strategy among possible defined strategies, the latter tend to be specified and used without guaranteeing their effectiveness [Naskos 2015a]. Therefore, SBP holders need means to assist them in defining and evaluating elasticity strategies before putting them in use in a real cloud environment.

The research conducted throughout this thesis has been motivated by those challenges which can be summarized in the following research question :

General Research Question :
How to assist in describing and evaluating elasticity strategies for Service-based Business Processes?

Hence, in this work, we particularly address the following research issues in defining and evaluating elasticity strategies for SBPs :

- ***Control System Decoupling*** :

  As mentioned in Section 1.1, elastic systems have a control system that manages the actual system through an elasticity controller designed to make elasticity decision and to perform system's adaptation/adjustment according to particular

elasticity information (*i.e.*, elasticity model and an elasticity strategy). This information is usually hard-coded in the controller. In the literature, as discussed in Chapter 3, some authors have proposed to decouple elasticity strategies from the controller to allow using different elasticity strategies and facilitate changing them. However, all the existing works were proposed based on a specific elasticity model restricting the definition and use of elasticity strategies to a particular elasticity capabilities and QoS metrics. Since elastic systems (SBPs) might have different control systems exposing different elasticity capabilities and different elasticity behavior, SBP holders need to be able to define and evaluate elasticity strategies regardless of a particular elasticity model. Thus, beside decoupling elasticity strategies from the controller, elasticity strategy specification should also be decoupled from any particular elasticity model.

- ***Elastic data-aware SBPs***:

  With the proliferation of the use of mobile devices and sensor-equipped environments, such as smart buildings, social media, and financial markets, an overwhelming "data deluge" is being streamingly produced every second around the world. This massive amount of data is a valuable asset in our today's information society. To harvest the valuable information hidden in these "data deluge", the concept of business processes is currently applied to streamline data processing and analytical steps [Dustdar 2011]. Therefore, resource-intensive tasks/services are no long part of only Scientific Workflows [Hoffa 2008, Juve 2010], but occur now in all kind of industries that have to handle and process large amounts of data in short time under specified QoS requirements. In such case, tasks requests are heavily different in the sense that some requests can be more data-intense than others and therefore have different performance expectation and require different QoS requirements. We believe that the data dimension of today's SBPs should have a particular attention during designing elastic SBPs and defining their elasticity information. Although some research works exist on elastic SBPs management, they do not consider the data-oriented characteristic of services requests and their distinguish requirements.

- ***Specification & Evaluation Support***:

  As previously mentioned, elasticity strategies as well as elasticity models are usually hard-coded in the controllers which makes them difficult to define and to maintain. The elastic SBP developers have to spend a lot of time and effort in putting together the control system (*i.e.*, elasticity controller, elasticity strategies and developing the logic of the elasticity capabilities that ensures the adaptation of the managed SBPs, *etc.*) and the evaluation system (using methods and techniques that require deep knowledge on them) rather than focusing on developing the business logic of the managed SBPs. To the best on our knowledge, there is a lack of comprehensive supports for defining and evaluating elasticity strategies

for SBPs that facilitate the task of IT managers/developers of elastic SBPs.

1.3 Motivating example

We provide here a simplified example of an elastic system for a data-aware process to motivate this thesis and to use it as a running example throughout this manuscript. We consider a data-aware process for molecular evolution reconstruction (MER) based on an input protein sequence of genomes [Ocaña 2012]. As illustrated in Figure 1.2, the MER process is composed of 8 services. The first service performs the pre-processing of FASTA file which contains a representation of nucleotide or peptide sequences. Its time complexity is O(NxL), where N is the number of sequences in the input file and L is the length of each sequence. The second service consists in constructing a Multi Sequence Alignment (MSA) using a program with time complexity $O(N^2 \times L) + O(N \times L^2)$ [Katoh 2008]. Thereafter, the third service converts the generated MSA in FASTA format to that in PHYLIP format. This conversion is performed with time complexity of O(N) where N is the number of entries in the file. Then, the fourth service which consists in pre-processing PHYLIP file formats the input file according to the format definition and generates a second PHYLIP file. This service is simply performed with a time complexity of O(N). After that, the fifth service receives the PHYLIP file as input and produces a phylogenetic tree as output [Stamatakis 2014]. It is of time complexity O(N^2). The constructed phylogenetic tree is used for the MER exploration. We consider in this example that the latter is performed by two parallel services (*i.e.*, 6.1 and 6.2) which output a set of files containing evolutionary information. Each of the services executes different phase of MER exploration in time complexity O(N^3) [Jansen 2001]. The last service processes the evolutionary information resulted from the previous services. This service is of time complexity O(N) where N is the summation of sizes of the output files resulted from services $S6.1$ and $S6.2$.

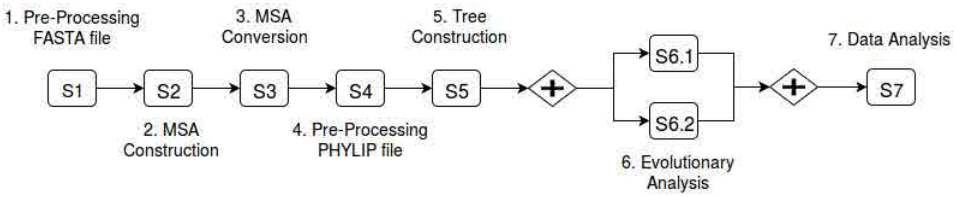


Figure 1.2: BPMN model of Molecular Evolution Reconstruction Process

As we can see, some services (*e.g.* S2) are more complex than others which make them expensive in terms of time and resources consumption. Their time and resources consumption are strongly related to the size of the request. Let's consider two sets of protein sequences of genomes P_1 and P_2. P_1 contains 5000 sequences of length 25000 while P_2 contains 10000 sequences of length 200000. The execution of S2 for P_1 and

P_2 will be performed respectively in 36 seconds and 65 minutes using a processor type Intel Core i7-3770k at 3.9 GHZ frequency which performs 106 924 MIPS (Millions of Instructions Per Second). The gap in execution time of P_1 and P_2 is very large, so would be the difference in users' expectations. Consequently, it is important to note that the characteristics of SBP requests should be considered when specifying QoS requirements and elasticity strategies.

We assume that at some point of time the process will be invoked many times with different input files which leads to overcome the composed services capacity which in turn leads to loss of QoS. Some input files will contain small sets of protein sequences of genomes that are expected to be processed in short time and to use a little amount of computational resources, while others will contain large sets that are expected to take much more time to be processed and its processing occupies a large amount of computational resources. For instance, the waiting for heavy requests to finish leads to unacceptable increase in the response time of light requests which in turn leads to disappointing users' expectations. So, we need to make sure that the process will keep running as expected and that all the requests will be processed while maintaining the desired QoS and satisfying users' expectations.

In order to prevent QoS degradation, we have to first specify the elasticity information (*i.e.*, the elasticity model and the elasticity strategy) on which the design and the functionality of the elasticity controller will be based. Different elasticity models can be specified describing different monitoring metrics and elasticity capabilities (*e.g.*, changing the characteristics/properties of the used resources, replicating the process/service, *etc.*) for ensuring the process adjustment to preserve the required QoS. For example, one may choose to maintain the required QoS by allowing the controller to duplicate/-consolidate the whole process (*i.e.*, all the services that compose the process) as many times as needed. So, he would specify an elasticity model describing two elasticity actions, *i.e.*, duplication and consolidation actions, to be implemented by the controller and applied for ensuring the process elasticity at process level. However, duplicating the whole process, while there are only few services, such as the service S2, requiring more capacity to serve the incoming requests, creates an over-provisioning of resources.

As the QoS violations at some point of time may come from one particular service, *e.g.*, service S2, and one particular category of requests, *e.g.*, light requests that are waiting for the heavy ones to finish, we thing it would be a better choice if the controller actions that will be performed to avoid such violations will be applied only on the bottleneck services while considering the characteristics of the requests causing the QoS degradation. Thus, we consider specifying an elasticity model for hybrid scaling that performs two main elasticity actions, namely *Duplicate* and *Consolidate*. The duplication action allows to add new service copy with different configuration. The service capacity and request groups property of a service are re-configurable by the action. The consolidation action allows to release service copies whenever needed.

To decide when, where and how these actions should be applied on our MER process, an elasticity strategy should be defined to be used by the controller. Many strategies

can be defined to provide better resource consumption and preserve the required QoS level using different metrics, rules composition, thresholds, *etc.* We consider, in our elastic system, the use of a reactive strategy that defines rules for the actions specified in the elasticity model based on the waiting time of requests. So, the duplication action creates a new copy of a service for requests under a specific group if at least one of waiting requests of that group exceeds the maximum waiting time threshold and the same applied for all its copies. Otherwise, if there is no longer waiting requests and the consumed capacity of a service copy is equal to 0 and the response time of the service is below its minimum threshold, a consolidate action is triggered by releasing the service copy. Here, nothing ensures the effectiveness of this strategy or the consistency between its rules if not carefully implemented/defined.

1.4 Scientific Contributions

The main objective of this thesis is to provide SBP holders a support for describing and evaluating their elasticity strategies to guarantee their effectiveness by detecting any suspicious behavior before investing in using them in the cloud. Defining a specific elasticity strategy or proposing resources allocation and scheduling algorithms for elastic SBPs are out of the scope of this thesis. We are mainly focusing on providing a support to the configuration phase in business process life-cycle based on petri-nets to allow the evaluation of strategies. In the light of the aforementioned problems, the main contributions that will be thoroughly detailed in the next chapters are the following.

Contribution I
Modeling data-aware elastic SBPs

Evaluating elasticity strategies for SBPs before using them in the cloud requires employing formal methods to describe their elastic execution environment which provide rigorous description and allows analysing elasticity behavior and guarantee their effectiveness. In our work, we employ High-level Petri Nets as a basis for this contribution which can be summarized as follows:

- We propose a data-aware formal model for describing elastic execution environment of SBPs based on high-Level Petri Nets. The model describes the characteristics of service engines, hosting services in a SBP, and their requests, by allowing to introduce the features that characterize them and distinguish them from each other, which makes it possible to define more sophisticated elasticity strategies using different elasticity indicators.

- We define and formalize two elasticity mechanisms/capabilities that allow performing hybrid scaling on the model by adding/removing service engine copies

with different configurations. The use of hybrid scaling method permits to customize the provided resources according to service's requests characteristics. Applying the defined elasticity mechanisms on our formal model according to a specific elasticity strategy changes its structure and status allowing the evaluation of the elasticity strategy to ensure its effectiveness before using it in a real Cloud environment.

A detailed description is given in Chapter 4. This contribution was originally presented in [Jrad 2017a].

Contribution II
Elasticity Model Description Language

An elastic system is usually managed by a controller that implements a specific elasticity model and uses an elasticity strategy to control the system adjustment decisions. In the context of elastic SBPs, an elasticity model defines the ground terms and functionalities that describe SBPs elasticity such as the elasticity capabilities/actions to be undertaken, metrics to monitor to trigger the actions and properties to access and reconfigure. It is the basis for specifying elasticity strategies and constructing an elasticity controller that manages and evaluates the elasticity of SBPs. Most of the existing works for providing elastic systems have been proposed based on a specific elasticity model that allows either vertical or horizontal elasticity. In order to provide a generic solution for describing and evaluating elasticity strategies for SBPs, we propose a domain-specific language, named StratModel, that allows :

- Describing different elasticity models, with different elasticity mechanisms/capabilities and customized monitoring metrics;
- Generating elasticity controllers, for a given elasticity model, that can be used to evaluate elasticity strategies for a given SBP model.

The elasticity capabilities in StratModel is defined by providing their mechanisms through a set of examples which illustrate how the generated controller should operate when applying an action that changes the structure of the managed SBP model. The language allows to generate elasticity controllers using a pre-defined template described using high-level petri net to allow the formal evaluation of strategies. A detailed description is given in Chapter 5. This contribution was originally presented in [Jrad 2017b]

Contribution III
A language for describing elasticity strategies

As mentioned in the previous section, defining non-trivial elasticity strategies that go beyond the set of features provided by existing solutions is still a challenging task. In

our work, we proposed a rule-based domain specific language, called STRAT, for specifying elasticity strategies governing SBP elasticity. An elasticity strategy is specified for/based on a specific elasticity model that provides the elasticity capabilities that can be applied on SBPs to ensure their elasticity. During designing STRAT language, two solutions have been in mind to make STRAT generic enough to describe strategies for different elasticity models which are as follows :

- The first solution was to provide STRAT language grammar with a constant set of actions used in the commercial cloud-solutions and the research papers. Such solution makes the users (SBPs holders) constrained to a set of pre-defined actions and parameters and does not enable the adaptation to new elasticity capabilities that can be provided by the community;
- The second solution, that we adopt, consists in linking STRAT to STRATModel (*i.e.*, contribution II) that allows users to separately provide the description of an elasticity model on which STRAT language will be based. So, the latter will provide users with only the elasticity capabilities defined in the elasticity model when defining their strategies.

We should note here that we use the terms "users" and "SBP holders" in the rest of this manuscript interchangeably to refer to the users of our solutions who can be anyone responsible for the configuration phase of elastic SBP in the cloud, such as IT manager. A detailed description of our language is given in Chapter 5. This contribution was originally presented in [Jrad 2016], and extended in [Jrad 2017b].

Contribution IV
Elasticity Strategy Evaluation Framework for elastic SBPs

Many elasticity strategies can be defined to steer SBPs elasticity. The abundance of possible strategies requires their evaluation in order to guarantee their effectiveness before using them in real Cloud environments. Based on the previous contributions, we introduce a framework, called STRATFram, for describing and evaluating, through simulation, different elasticity strategies for elastic SBPs. The framework allows users (SBP holders) to first define different elements of the elastic systems they want to evaluate their elasticity behavior, which are typically comprised of :

1. An elasticity model on which the strategies will be based;
2. A SBP model for which the strategies will be defined;
3. Elasticity strategies that will be evaluated;
4. A simulation configuration that specifies the needed parameters for the purpose of the evaluation.

StratFram provides a set of languages, including StratModel (*i.e*, contribution II) and Strat (*i.e.*, contribution III), designed to generalize the use of the framework and to facilitate the description of each element. The evaluation of elasticity strategies using StratFram consists in providing a set of plots that allows the analysis and the comparison of strategies to choose the ones that fit to the target SBP and usage behaviors. A detailed description of our framework is given in Chapter 6. This contribution was presented in [Jrad 2019].

1.5 Thesis Structure

The remainder of this manuscript is organized as follows.

Chapter 2: Cloud Computing & Business Process

This chapter briefly introduces the concepts of cloud computing and business processes that are needed to grasp the remainder of this thesis. In the first part, we present the definitions of Cloud computing and its fundamental characteristics. Particularly, we focus on defining the principles and characteristics of one of the most important cloud computing feature which is *Elasticity*. The second part presents how the research and industry communities have viewed and defined business processes and what phases a business process goes through during its life-cycle. The final part of this chapter gives a short introduction to the concept of elastic service-based business processes (elastic SBPs), or simply *elastic processes*.

Chapter 3: State of the Art

This chapter is devoted to review existing literature related to the present research. We first study existing works proposing elasticity models and mechanisms for ensuring elasticity of cloud systems. Then, we survey existing languages for describing elasticity strategies as well as works on elasticity strategies evaluation. The final part of this chapter gives a synthesis of the studied works.

Chapter 4: Modeling of Data-aware Elastic SBPs

In this chapter, we present our proposal for modeling elastic execution environment of data-centric processes. The proposed model is defined based on high-level petri nets. It permits to describe the characteristics of service engines hosting SBPs services and their requests. For better understanding, we start the chapter with a brief introduction to some basic mathematical notations and petri net related concepts. Thereafter, we detail our formal model that intertwines two elasticity mechanisms for ensuring hybrid elasticity of SBPs at the service level.

Chapter 5: Description of Elasticity Strategies for Elastic SBPs

This chapter is devoted to present our contributions (II) and (III), namely StratModel language for describing elasticity models and Strat language for describing elasticity strategies for elastic SBPs for a given elasticity model defined using StratModel. We start the chapter with a short introduction to the concept of domain-specific language followed by an overview of languages design. Then, we present the details of StratModel language and Strat language with their use in defining respectively the elasticity model and the elasticity strategy of the motivation example.

Chapter 6: Elasticity Strategies Evaluation Framework

In this chapter, we present a framework, named StratFram, for describing and evaluating elasticity strategies for elastic SBPs. It enables the evaluation, through simulation, different elasticity strategies for different elasticity models based on our formal model of elastic execution environment of SBPs. The chapter is divided into two parts. In the first part, we introduce our framework architecture and describe its components. Then, in the second part, we detail the implementation aspects and the evaluation scenarios as well as their results.

Chapter 7: Conclusion and Perspectives

This chapter concludes our thesis by summarizing the contributions. We also present in this chapter our research perspectives that we aim to realize at short and long term.

CHAPTER 2

Cloud Computing & Business Process

Contents

2.1 Introduction

In this chapter, we present background knowledge (basic concepts) required for the understanding of the remainder of this thesis. The chapter is divided into three essential parts. The first part covers the definitions and fundamental characteristics of Cloud Computing, in general, and its elasticity property, in particular. Afterwards, we introduce in the second part what exactly is a business process according to the research and industry community. Finally, we present the integration of Business process and Cloud Computing which emerged the so-called Elastic SBP.

2.2 Cloud Computing

2.2.1 What is a Cloud Computing ?

"*Cloud computing is 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*" [Hogan 2011]. This is how the National Institute of Standards and Technology (NIST) defined the cloud computing. It has been considered as the best definition among several other definitions abound for cloud computing from both academia and industry [Ruparelia 2016]. Some of the other definitions are listed to give an idea of how the notion of '*cloud computing*' has been viewed by researchers and industry analysts.

> "*A computing Cloud is a set of network enabled services, providing scalable, Quality of Service (QoS) guaranteed, normally customized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way*", [Wang 2008].

> "*Building on compute and storage virtualization technologies, and leveraging the modern Web, Cloud computing provides scalable and affordable compute utilities as on-demand services with variable pricing schemes, enabling a new consumer mass market*", [Klems 2009].

> "*Cloud Computing is a style of computing where scalable and elastic IT-related capabilities are provided as a service to external customers using internet technologies*", [Cearley 2010].

> "*Cloud computing is a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLA's*", [Vaquero 2009].

> "*Cloud Computing is an emerging IT development, deployment and delivery model, enabling real-time delivery of products, services and solutions over the Internet (i.e., enabling cloud services)*", [Gens 2008].

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). Figure 2.1 shows the service models of the cloud as viewed by NIST and the general requirements and processes for cloud providers to build each of them.

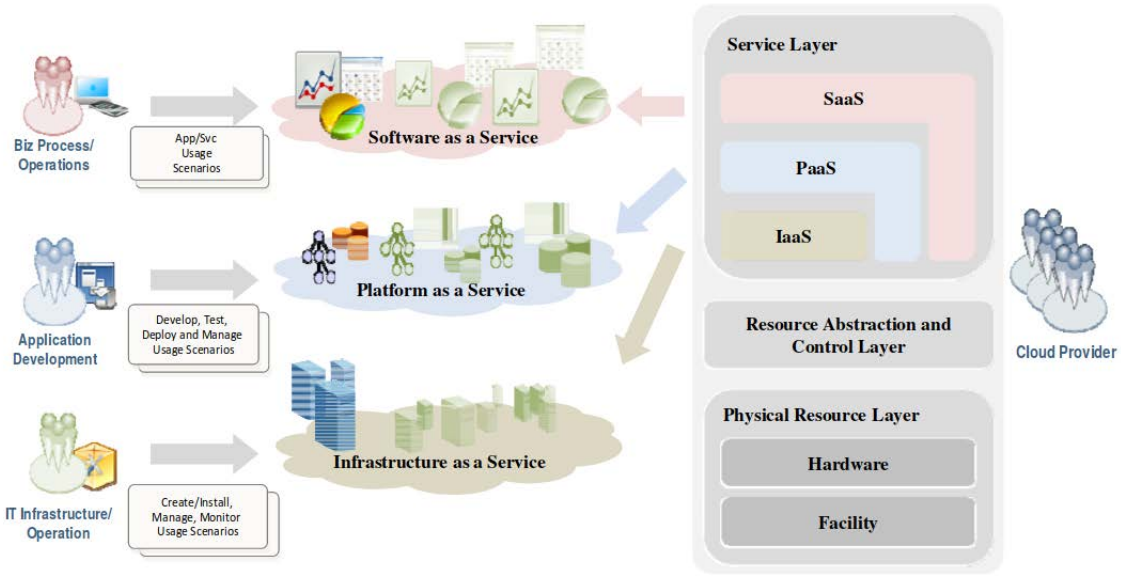


Figure 2.1: Service models of Cloud computing [Hogan 2011]

Software-as-a-Service (SaaS) : Consumers are offered to use running applications deployed on a cloud infrastructure as long as they want and pay for what they use. They are not responsible for managing or controlling the underlying cloud resources or even individual application capabilities. The most widely used SaaS is Google Apps.

Platform-as-a-Service (PaaS): Consumers are provided with a plaform that includes programming languages, libraries, services and tools needed to deploy and run consumer-created or acquired applications in the cloud infrastructure. The consumer is only responsible on controlling the deployed applications and optionally configuration settings of the environment on which the application is hosted. Google AppEngine and Microsoft Azure are well-known examples of PaaS.

Infrastructure-as-a-Service (IaaS): Consumers are provided with the capability of provisioning computing resources on which they can deploy and run any software from operating systems to applications. The consumer is not responsible for controlling and managing the underlying cloud infrastructure. Amazon EC2 is a key example of IaaS.

The building of the three cloud service models can be made either on top of one another (*i.e.*, SaaS built upon PaaS and PaaS built upon IaaS) or directly upon the underlying cloud resources. For example, a SaaS application can be implemented and hosted on virtual machines running in IaaS or directly on top of physical cloud resources without using IaaS [Hogan 2011].

Nowadays, more services have appeared targeting a specific area called generally as XaaS. For example, there is the NaaS for Network as a Service, DaaS for Desktop as a Service, etc.

2.2.2 Characteristics of Cloud computing

The Cloud Computing has been recognized for its features that distinguish it from other computing paradigms, like Grid Computing. These features can be summarized in the following aspects [Wang 2008, Hogan 2011]:

- **On-demand self-service**: A consumer is able to provision computing resources on demand whenever needed in a simple and flexible way without requiring human intervention on the service provider's side.
- **Quality Of Service (QoS) guaranteed offer** : Cloud computing provides consumers with computing environments that can guarantee the required QoS such as memory size, availability, response time, service throughput, *etc.* that are generally provided in Service Level Agreement (SLA). The latter is a negotiation between the service consumer and the service provider on the levels of availability, performance, operation, billing and even penalties in the case of violation of the SLA, *etc.*;
- **Resource pooling** : Cloud providers are allowed to pool large scale computing resources to serve multiple consumers. Different physical and virtual cloud resources are dynamically assigned and reassigned according to consumer demand.
- **Rapid elasticity**: Cloud computing resources can be dynamically provisioned for cloud services on demand whenever needed and released when they are no longer needed. They appear to consumers that they are unlimited and can be taken in any quantity at any time.
- **Measured service** : Cloud services are monitored and controlled for variour reasons, including effective resources utilization, billing or overall predictive planning. This feature provides transparency for both the service provider and consumer.

In the following section, we describe in more details what is the elasticity and which are the characteristics of the elasticity in the cloud.

2.2.3 Elasticity

2.2.3.1 Elasticity in physics

Elasticity is the ability of an object to resist a distorting influence and to return to its original size and shape after that influence or force is removed. When appropriate forces are applied on solid objects, they will deform and change their shape/size. If

their material is elastic, they will return to their initial size and shape when the applied forces are removed [Shawky 2012].

2.2.3.2 Elasticity in the cloud

Elasticity is the most important feature of cloud computing which distinguishes it from the other computing paradigms, *e.g.*, cluster and grid computing. It is considered as the primary reason behind choosing many companies to move their operations to the cloud and to adopt the cloud-based technologies in their day-to-day activities. Many researchers have attempted, for years, to define the cloud computing elasticity. Despite that, there is no consensus on a common definition of the term elasticity in the cloud computing context [Bikas 2016]. Here, we list some of the commonly used definitions of cloud elasticity to get the perspective about different usages of this term.

> "*Computing resources can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time*", NIST's definition of Elasticity [Mell 2011].

> "*Elasticity is the degree to which a system is able to adapt to workload changes by (de)allocating resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible*" [Herbst 2013].

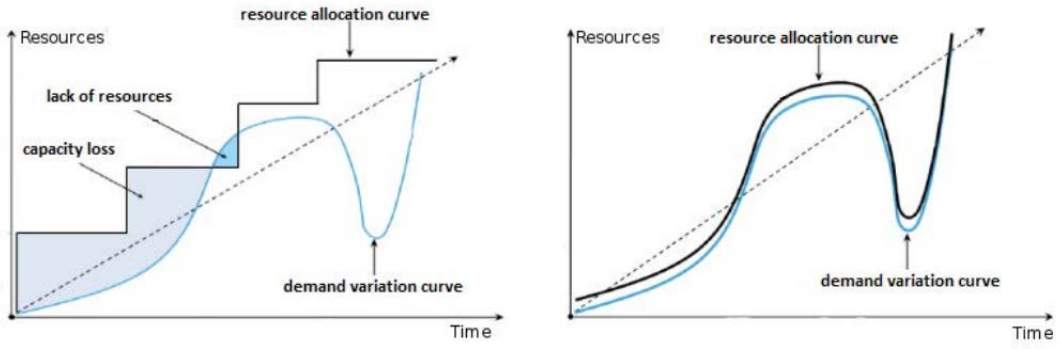
> "*Ability of the system to adaptively scale resources up and down in order to meet varying application demand*" [Han 2014].

> "*How quickly a system can adapt to changes in the workload that may happen in a short amount of time.*" [Li 2011].

> "*How much a cloud service can be scaled up and down during the peak times*" [Garg 2013].

> "*Capacity at runtime by adding and removing virtual resources without service interruption in order to handle variation in the workload*" [Perez-Sorrosal 2011].

> "*Elasticity is basically a 'rename' of scalability, which has been a known non-functional requirement in IT architectures for many years already. Scalability is the ability to add or remove capacity, mostly processing, memory, or both, to or from an IT environment when this is needed,*" as defined by Edwin Schouten, IBM [Schouten 2012].



(a) (b)

Figure 2.2: The principle of elasticity

In this thesis, we consider the elasticity as the ability of a system to be adjustable to the variation in the workload by provisioning as many resources as required in autonomic manner in order to meet the QoS requirements. Traditionally, the resource allocation has been done by provisioning a fixed amount of IT resources that could handle high demands at peak periods. This approach is not suitable to be used in high dynamic environments such as Cloud environments. Acquiring a fixed amount of resources would lead to an over-provisioning of resources at the majority of period of time and therefore to loss of money. Also, when demands gets higher than expected in peak periods, the fixed allocated resources will not be sufficient to handle them which would lead to under-provisioning of resources and therefore to violate QoS requirements. Ensuring elasticity makes it possible to have a resource provisioning that automatically adjusts to workload changes in order to meet QoS requirements while reducing cost.

Let's take the resource demands shown in Figure 2.2, the used resource allocation approach should ensure the provisioning of sufficient resources that can satisfy the demands at any period of time while preventing the under-utilization of resources. Figure 2.2(a) illustrates the use of traditional resource allocation approach. As illustrated, this approach causes an over-utilization of resources most of the time and an under-utilisation of resources when the demands get higher than the capacity of the already allocated resources. In this case, the best solution as shown in Figure 2.2(b) is to ensure the dynamic allocation of resources that automatically adjusts to demands changes while ensuring QoS and reducing cost.

2.2.3.3 Characteristics of elasticity in the cloud

The elasticity can be described based on four characteristics as illustrated in Figure 2.3: (1) Scope, (2) Method/Technique, (3) Metric, and (4) Strategy [Galante 2012].

1. **Scope** : Scope defines the cloud level in which the elasticity actions are applied, *i.e.*, infrastructure level (IaaS clouds), platform level (PaaS clouds) or service level

(SaaS). Generally, elasticity actions (supported by underlying infrastructure) are trigged by an elasticity controller that is either provided by IaaS clouds or embedded in the application or within the execution environment (called containers) used by PaaS clouds to automatically manage the resources used by application;

2. **Method/Technique** : An elasticity method/technique refers to the method of provisioning resources employed in the elasticity solutions. Provisioning of resources can be made using either horizontal elasticity, vertical elasticity, or hybrid elasticity (combination of horizontal and vertical elasticity) (*cf.*, Figure 2.4). Horizontal elasticity consists in adding/removing instances of system elements to balance the current workload. It is also known as replication or duplication/consolidation of resources. On the other hand, vertical elasticity consists in changing the properties/characteristics (*e.g.*, CPU cores, memory) of the used instances in the system by increasing or decreasing them. It is also known as resizing of resources. As a combination of vertical and horizontal elasticity, hybrid elasticity allows the adding/removing of instances with different characteristics to/from the systems. Along these basics methods of elasticity, another method, named Migration, is used to allow different way of resource resizing (vertical elasticity). It consists in transferring an instance that is running on a specific physical resource to another one that might have different properties/characteristics (*e.g.*, more memory or less memory) and best fits to the system load. These elasticity capabilities/methods are the main constructions of an elasticity model that defines the ground terms and functionalities/features supported by the target cloud environment and used to describe the elasticity of the managed system such as the elasticity actions to be undertaken, metrics to monitor to trigger the elasticity actions and properties to access and reconfigure.

3. **Metric** : A metric represents an elasticity indicator that is monitored and used for making elasticity decisions. There are generally four categories of metrics used in the literature [Ameller 227, Emeakaroha 2010, Heidari 2014, Frey 2013]: (1) Instance-based metrics such as CPU utilization, memory utilization, and Surge Queue length (a count of total number of requests that are waiting to be submitter to a target resource); (2) Service-level metrics such as service availability, service response time, and service throughput; (3) Network-based metrics such as network utilization and network latency; and (4) elasticity constraints such as task/process deadline and cost/price;

4. **Strategy** : An elasticity strategy is a policy used to make elasticity decisions. There are two common elasticity strategies: manual and automatic. Manual strategies mean that the user is responsible for monitoring his/her deployed application and for performing elasticity actions whenever needed through a public interface provided by the cloud provider. On the other hand, automatic strategies mean that the control and the elasticity actions are taken by an elasticity com-

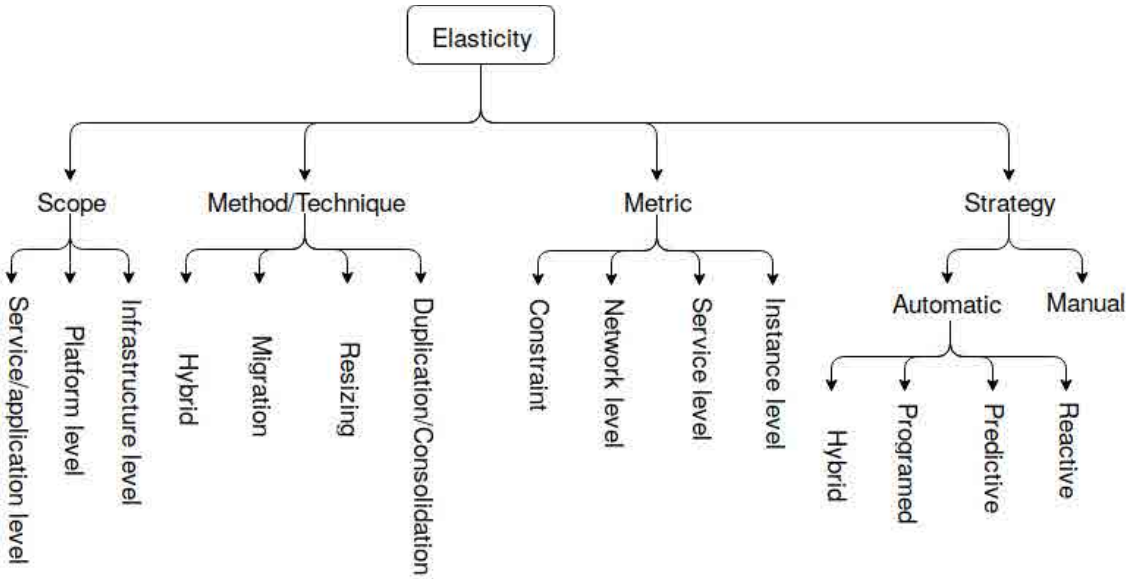
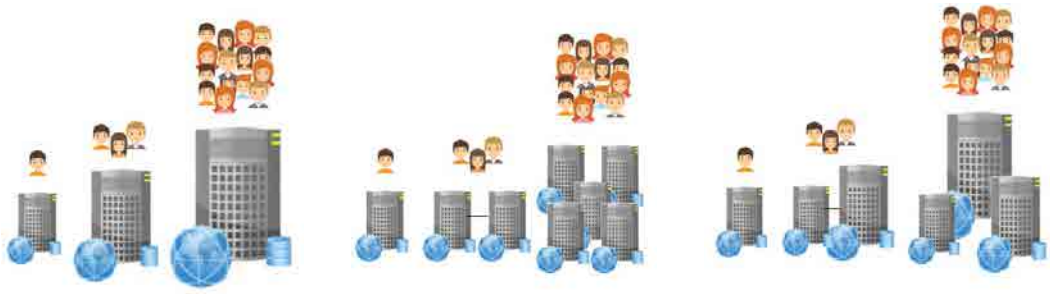


Figure 2.3: Elasticity Characteristics

ponent (*e.g.*, elasticity controller) provided by the cloud system or embedded in the application or within the execution environment. The elasticity component is responsible for monitoring the deployed application, collecting information related to its execution, and performing elasticity actions in accordance with user-defined rules such as the ones specified in the Service-Level Agreement (SLA). An elasticity strategy can be either (*i*) reactive, (*ii*) predictive, (*iii*) programmed, or (*iv*) hybrid.

- A reactive strategy is a rule-based strategy in which a rule is in the form of Rule-Condition-Action. A rule is composed of a set of conditions and an action to be performed. An action is triggered when specific conditions are satisfied. A condition considers either an event or a metric of the system. It is generally a threshold-based condition. So, once a threshold is violated, an action will be performed.
- A predictive strategy is a policy that uses heuristics and analytic techniques such as predictive performance models and load forecasts to anticipate the future system violations and to decide how and when to perform elasticity actions in order to prevent these violations.
- A programmed strategy is based on scheduling the execution of an elasticity action at a specific period of time or after a period of time (*e.g.*, increase the number of resources at the peak period).
- A hybrid strategy is a combination of using Rule-Condition-Action mecha-



(a) (b) (c)

Figure 2.4: The different type of resources elasticity: (a) Vertical Elasticity, (b) Horizontal Elasticity, (c) Hybrid Elasticity.

nism with some heuristics and analytic techniques and some scheduled actions.

2.3 Business Processes

2.3.1 What is a Business process?

A business process is a set of one or more linked activities/steps performed collectively by a group of stakeholders in order to achieve a business objective or policy goal, within the context of an organizational structure defining functional roles and relationships. Each activity/step in a business process denotes a task that is completely performed by either a human actor or an IT system. A process can be fully contained within a single organizational unit or can extend across several different organizations such as in a customer-supplier relationship. It is the fundamental building block for many related ideas such as Business Process Management (BMP), Process Automation, *etc.* As with most of terms, many researchers have tried over the years to define what exactly is a business process. In the following, we provide the most common definitions found in the literature.

> "A business process is a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. A business process has a goal and is affected by events occurring in the external world or in other processes" [Hammer 1993].

> "A business process is simply a structured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organisation, in contrast to a

> product's focus on what. A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action" [Davenport 1993].
>
> "A business process is the set of internal activities performed to serve a customer" [Jacobson 1994].
>
> "A Business process is a lateral or horizontal organisational form that encapsulates the interdependence of tasks, roles, people, departments and functions required to provide a customer with a product or a service" [Earl 1994].
>
> "Business process is a purposeful activity carried out collaboratively by a group, often crossing functional boundaries and invariably driven by outside agents or customers" [Ould 1995].

In this thesis, we consider a special case of business processes, named Service-based Business processes (SBP). A SBP is a business process that is composed of only a set of elementary IT-enabled services related to each other according to their contribution to the overall goal of the process. Typically, a service refers to the smallest unit of work representing a module that offers computational or data capabilities. Services composing a SBP carry out specific business activities and they can be assembled to form the SBP via any appropriate service composition specifications such as Business Process Execution Language (BPEL) [Juric 2006], Business Process Modeling Notation (BPMN) [Allweyer 2010], *etc.*

2.3.2 Business process life-cycle

A business process goes through four main phases related to each other representing its life-cycle: (*i*) Modeling (also referred to Design) & Analysis, (*ii*) Configuration, (*iii*) Enactment, and (*iv*) Evaluation [Weske 2012] (*cf.*, Figure 2.5). These phases are organized as a loop structure showing their logical dependencies and allowing a continuous evolution of the business process.

Modeling & Analysis : Business process modeling is the first and fundamental step in a business process life cycle. It allows companies to identify and validate their processes. So, after determining each component that can contribute in the construction of business processes from the set of activities that compose them to the roles and the link between them, a graphical notation is used to express the processes and produce the so-called a business process model. Such models are used as a communication basis for different BP holders and aid at refining and improving processes. They provide BP holders the process logic independently of a possible IT infrastructure. In order to ensure their correctness, the described process models are also analysed and verified within this phases to provide a well described models.

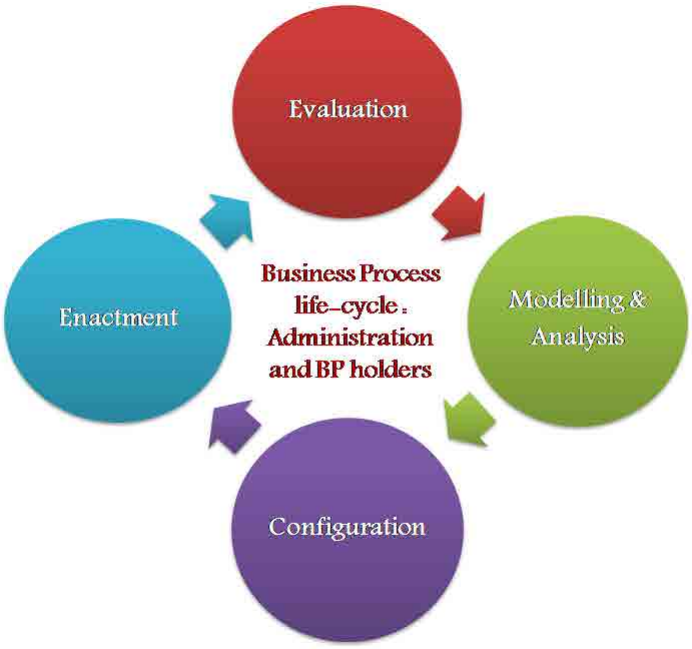


Figure 2.5: Business process life-cycle

Configuration : The objective of this phases is to enhance the described business process models with technical information that facilitates the enactment of the process by a dedicated business process management system. So, the process management system and IT system have to be configured according to the organizational environments of the company. Thus, this configuration phase outputs process models that are configured for a particular enactment infrastructure.

Enactment : After completed the configuration phase, instances of the configured processes can be enacted. They are executed within the enactment phase in order to fulfil particular business goals. During their execution, business process management system controls them in order to gather valuable execution information and visualize their status. At the end of this phase, several process instance have been performed and their execution information has been gathered in execution log files.

Evaluation : The evaluation phase is the last phase in business process life-cycle which uses the information gathered during the execution of process instances to evaluate and improve the original business process models and their implementations. So, the execution log files are used and evaluated using process mining techniques which aim at identifying the quality of process models and the adequacy of the execution environment.

As the work presented in this thesis targets one particular aspect of business process modeling and can be used during the configuration phase that is based on process models, we present in the next section more details on the process modeling.

2.3.3 Business process Modeling

Business process modeling, often known as process modeling, is the activity of providing an analytical representation (or illustration) of business processes of an organization. It allows organizations to visually document, analyse, improve and automate their processes. Process modeling is widely seen by business process community as a critical component for providing successful business process management (BPM).

A wide variety of graphical process modeling languages coming from different facets of scientific tradition have been used in research and industry to model and represent business processes. These languages can be categorized as formal, conceptual or execution languages. Formal languages are designed based on mathematical foundation. They are generally provided with unambiguous semantics for modeling the behavior of a system and allows analysis methods/techniques, such as model checking and simulation, to provide answers for questions related to correctness and performance. Petri Nets [Murata 1989] are one example of such languages. They are state-transition systems that are conventionally used to formally model concurrent systems. Their particularity is in having a mathematical definition of their execution semantics and a well-developed mathematical theory for process analysis.

Unlike formal languages, conceptual languages are typically informal and they lack a well-defined semantics and analysis capability. BPMN [Allweyer 2010] is the most used conceptual language for modeling business processes. It is considered as the world-wide recognized industry standard notation for specifying business processes. BPMN is viewed by many consultants as the "Rolls Royce" of process modeling languages since most of the other modeling languages have been developed for other purposes and then adapted for modeling business processes. Though, there are other analysts and consultants who prefer to use other languages/methods to model their processes depending on what they want to do with the models. Beside conceptual languages, there are execution languages which are "technical" languages used for specifying enactment of business processes. BPEL [Juric 2006] is the most known example of such languages. It is a standard executable language that can be used to specify business process behavior based on web services.

2.4 Business Processes in the Cloud

In today's industry, business processes, especially service-based business processes (SBPs), are getting more and more complex and resource-intensive. In fact, resource-intensive tasks are no long part of only Scientific Workflows [Hoffa 2008, Juve 2010], but occur now in all kind of industries that have to handle and process large amounts of data

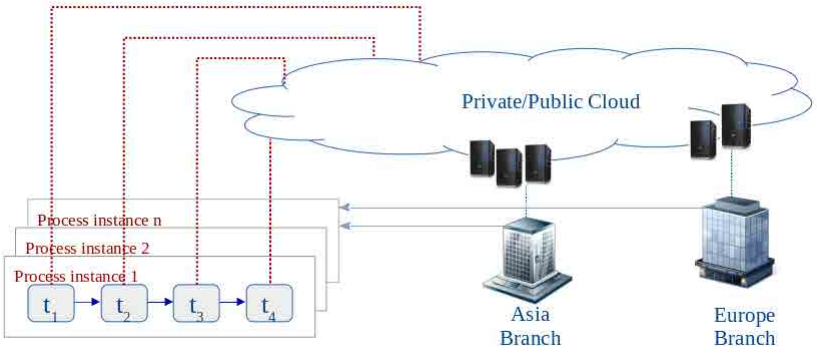


Figure 2.6: Service-based business process in the cloud - Elastic SBP

in short time. In fact, many industry domains and business areas, such as eHealth [Mans 2010], manufacturing [Schulte 2014], or SmartGrids [Rohjans 2012], feature extensive business process landscapes that are hard to predict essentially with the large number of changing business process requests/instances, which might occur in a regular as well as in an ad-hoc manner, and/or the need to process a massive amount of data in particular tasks. Especially when there is a large amount of data to be processed by some tasks in the process or having large number of process requests that occur in an ad-hoc manner, the process will rapidly change its resource requirements which might lead to a suboptimal usage of the available computational resources such as RAM or CPU. This may lead to either over-provisioning or under-provisioning of resources [Schulte 2014].

Cloud computing offers business process organizations a promising solution for managing the steadily increase of complexity in their business process executions and providing optimal usage of computational resources using its basic features such as the rapid elasticity (*cf.*, Section 2.2.2). In this context, a new research trend has emerged called Elastic SBPs (or elastic processes) [Dustdar 2011]. It refers to business processes that utilize cloud resources dynamically according to workload changes and then performing their tasks in an efficient and cost effective way. For instance, the concept of elastic SBPs can be realized in an industry domain like the financial industry. The latter can be considered as one of many other industry domains that feature extensive business process landscapes that need to be executed using cloud resources. Figure 2.6 illustrates a simplified service-based business process scenario from the banking industry deployed and running in the cloud. It describes a process of an international bank with branches in Asia and Europe. Each branch operates using a private/public cloud for data processing. Computational resources are needed to perform long-running data analytic processes as well as carrying out shorter processes such as the ones for trading or credit approval. However, the exact amount of the required computational resources is hard to predict since the number of process requests/instances and the amount of data that have to be handled concurrently can vary to a very large extent. So, in case of

over-provisioning of resources, the bank should lease more resources to extend the available computational resources. In such case, traditional BPMSs are no longer suitable. Hence, with the functionalities of BPMS, an elasticity controller is needed to automate the provisioning of resources by controlling when, where and how to provision and/or deprovision resources as provided by a given elasticity strategy in order to adjust them to the process/service workload.

2.5 Conclusion

In this chapter, we introduced the basic concepts related to the work presented in this thesis. We detailed in the first part the principles and characteristics of cloud computing focusing, in particular, on its elasticity property which is considered the most important feature behind its popularity. In the second part, we presented what is a business process from the perspective of industry and research communities as well as its life-cycle and particular its modeling. To conclude this chapter, we presented a newly emerged trend, named elastic processes, that combines the two presented concepts, *i.e.*, business process and cloud computing. The next chapter will discuss in more details the principle approaches that proposed elasticity models and mechanisms for elastic systems and the ones that especially targeted the definition and evaluation of elasticity strategies.

CHAPTER 3

State of the Art

Contents

3.1 Introduction

Since the advent of Cloud Computing, more and more companies are moving their business process to the Cloud. They are now required to manage their cloud services at any time to preserve their Quality of Service (QoS). The quality and reliability of the cloud services become an important aspect, as customers have no direct influence on services. QoS has been a critical issue in several customer-centric disciplines such as manufacturing ([Zhou 2009, Xu 2012]), healthcare ([Kenagy 1999, Ray 1999, Aktas 2015, K. 2015]) and information management ([Liu 2000, Smith 2005, Garcia-Recuero 2014, Sandhu 2015]). It denotes the levels of performance, reliability, and availability of a service/process offered by the platform or the infrastructure that hosts it. The expectation of cloud users from providers to deliver the required level of service quality and the neediness of cloud providers to find a good compromise between QoS levels and operational costs, are what make QoS a fundamental issue for both parties.

Elasticity played an important role in many research works that propose methods and mechanisms to harness the ability of services/processes running in the cloud to be elastic regarding the change in workload to ensure that the customer gets the desired level of QoS while avoiding over-provisioning and under-provisioning of resources. While

few works have tackled the problem of elastic processes in the cloud, there is a plethora of works dealing with ensuring applications elasticity in the cloud. In this chapter, we will discuss a selection of works dealing with providing efficient elastic systems in the cloud. We will start by presenting approaches that propose models and mechanisms for ensuring systems elasticity (*e.g.*, service, application, SBP, *etc.*) 3.2. Thereafter, we will present different languages for describing elasticity strategies in Section 3.3. In Section 3.4, we will present the state of the art of elasticity strategies evaluation. Finally, we will conclude this chapter by a synthesis of the presented works comparing them on the basis of our research objective in Section 3.5.

3.2 Models and mechanisms of elasticity

In this section, we will present a selection of works dealing with/considering elasticity models and mechanisms in cloud environments.

In [Bessai 2012b, Bessai 2012a], cloud computing's elasticity has been considered during tackling the problem of resource allocation and task scheduling for business processes. In their works, Bessai *et al.* proposed bi-criterion approaches based on two objective functions defined for the overall completion time and the execution cost of using a set of resources composed of virtual machines (VMs) and human resources. Their proposals were based on the assumption that the users dispose of an unbounded number of VMs that can be used to execute business process tasks while minimizing the overall completion time and/or the execution cost of executing the business process instances. The authors suppose that the users will manually lease and release VMs whenever needed in order to perform the task allocation and scheduling of tasks of business process instance with the illusion of having unbounded number of VMs. However, this requires effort from the users to monitor and control system state in order to lease (release) additional (unneeded) VMs in the right time. In this work, we are interested in automatically ensuring service-based business processes elasticity which can be performed by elasticity controllers based on elasticity strategies.

In [Hoenisch 2014], the authors have worked on providing elastic Business process management system (eBPMS) for the cloud. They presented a platform named ViePEP - the Vienna Platform for Elastic Processes which acts at the same time as BPMS and as an elasticity controller. ViePEP is proposed as a broker middleware that is responsible on receiving process requests sent by the clients and managing their execution in the cloud. It is designed according a control loop used in the field of autonomic computing named MAPE-K loop (*i.e.*, Monitor, Analyse, Plan, Execute, Knowledge) [Jacob 2004] in order to allow it to handle hundreds of process requests by ensuring the process elasticity while preserving the required QoS which can be provided for the process level or even the service level. The platform ensures the process elasticity by allowing the lease and release of VMs whenever needed and its stability by allowing the migration of a service deployed in one VM to another one. Figure 3.1 illustrates that ViePEP has five essential entities, *i.e.*, client, *BPMS VM*, *Backend VM*, service repository, and shared

memory. So, the client models his service-based process using a modeling API and optionally defines SLAs that provided the QoS requirements . The *BPMS VM* is the central component of ViePEP. It offers ViePEP core functionalities from receiving process requests, scheduling them according to given deadlines, and computing the amount of required resources based on an implicit predictive elasticity strategy for predicting future resources demands from historical data published in the shared memory allowing the reasoner component to decide on what action should be performed, *i.e.*, lease a VM, release a VM or move a running service to another VM, allowing then the horizontal scaling.

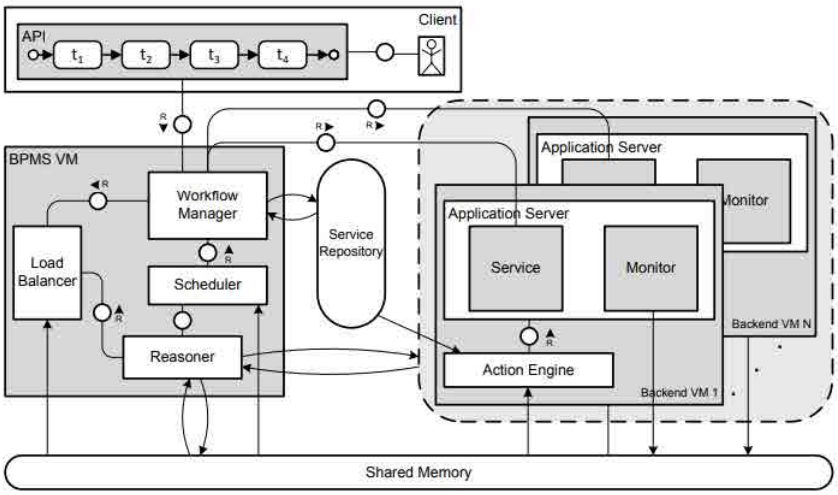


Figure 3.1: ViePEP architecture overview [Hoenisch 2014]

In [Ali-Eldin 2012], Ali-Eldin *et al.* proposed two adaptive horizontal elasticity controllers that control cloud services elasticity by adding and removing VMs whenever needed to preserve QoS requirements, *i.e.*, service capacity. The authors first proposed to model cloud services using queuing theory to allow the estimation of the future service load. Then, using the service model, the authors created two adaptive proactive elasticity controllers that base their decisions on the history of service load (*cf.*, Figure 3.2). These proactive controllers have been used to build an hybrid elasticity controller that consists of two controllers, one for deciding scale-out actions while the other for deciding scale-in actions. This hybrid controller is constructed to use both reactive and proactive controllers to dynamically change the number of VMs allocated to a cloud service. While the proactive controller made its decision based on the workload history, the reactive controller makes elasticity decisions based on the current workload.

In their work [Loff 2014], Loff et Garcia proposed a generic elasticity controller for cloud applications, named Vadara, that transparently connects and abstracts different existing cloud providers and enables to use different elasticity strategies defined

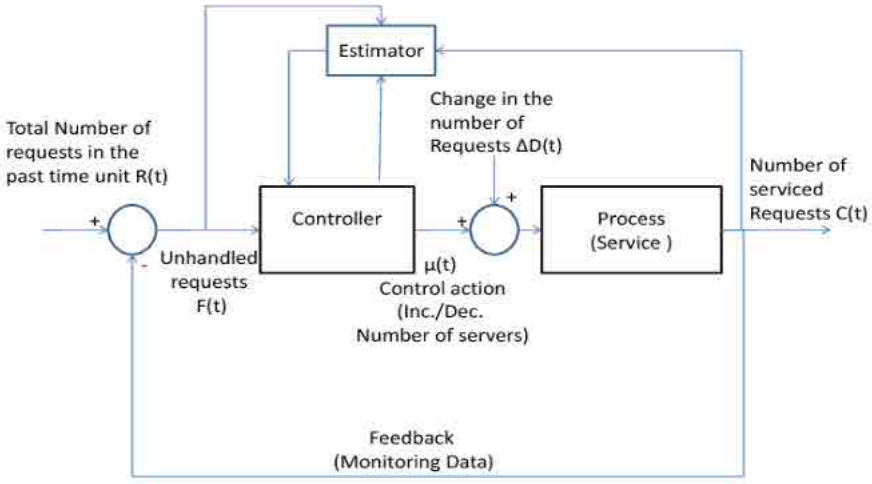


Figure 3.2: Horizontal Elasticity Controller Model[Ali-Eldin 2012]

in unified manner. Vadara designed based on an elasticity model for horizontal scaling allowing a cloud application running in a cloud provider to scale out/in when the controller detects any under-provisioning/over-provisioning of resources based on their usage such as CPU utilisation. The authors have proposed in their work to decouple elasticity strategies from their controller allowing then the use of different strategies for deciding whether to lease or release cloud provider's resources. In order to connect Vadara to a particular cloud provider and allow the management of cloud applications elasticity, an extension of Vadara abstract components should be provided according to the cloud provider specification. As shown in Figure 3.3, Vadara is composed of four main components and a repository component that stores any valuable information needed by the other components. The central component of Vadara is the core component which is responsible on configuring and initializing the other components including the cloud provider extensions. The monitor component is responsible on collecting, aggregating and sending monitoring metrics to the decider component to analyse them and decide on the appropriate elasticity actions that are therefore communicated to the scaler component. The scaler in Vadara is responsible on forwarding the received elasticity actions requests to providers scaling services in order to execute them.

A self-trained elasticity controller, called OnlineElastMan, has been proposed in [Liu 2016] for managing cloud-based storage systems elasticity. It has been defined to automatically train and evolve itself while serving workload. This self training allows the controller to update its control model that is used to make elasticity decisions for adding/removing servers to/from the underlying storage system to guarantee the required QoS, *i.e.*, required level of percentile latency. As illustrated in Figure 3.4, OnlineElastMan is composed of (*i*) a elasticity controller, (*ii*) an online model training

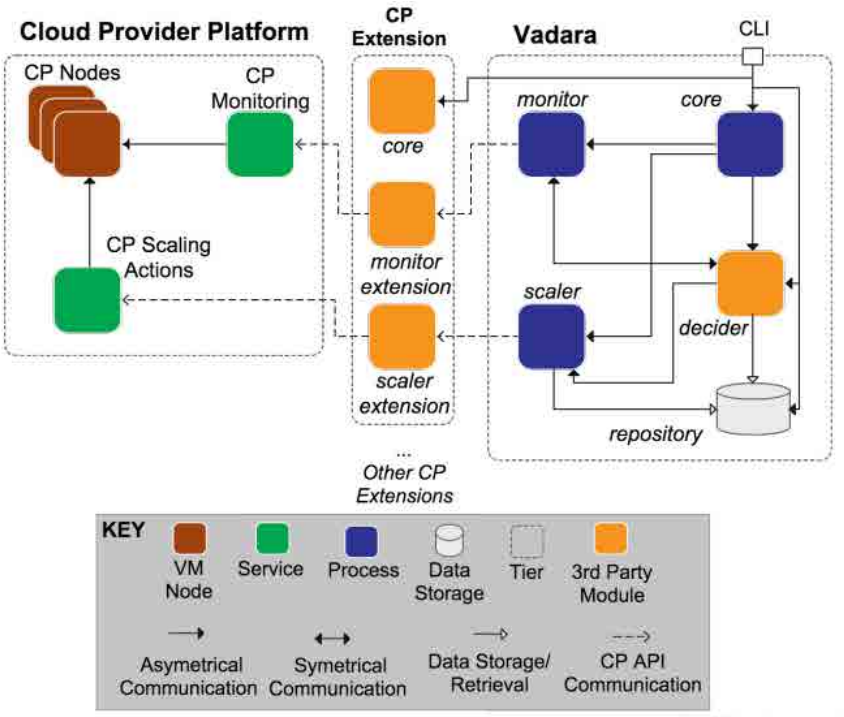


Figure 3.3: Vadara architecture overview [Loff 2014]

module, and (*iii*) optionally a workload prediction module that allows to predict in advance the future system workload based on patterns found in the current workload using an algorithm based on the regression trees. The elasticity controller is responsible of making scaling decisions for adding or removing servers/VMs for the underlying storage system by analysing the predicted future workload received from the workload prediction module and the multi-dimensional performance model sent by the online model training module. The latter is designed as an online model that automatically evolves and updates periodically using SVM (Support vector machine) model training technique to adapt to execution environment changes.

In their work [Farokhi 2016], Farokhi *et al.* have proposed a hybrid elasticity approach that uses control theory to synthesize a controller, named hybrid memory controller, for vertical memory elasticity of cloud applications. The controller is designed following a Monitoring, Analysis, Planning and Execution (MAPE) loop for autonomic computing [IBM 2005]. As shown in Figure 3.5, the monitoring phase consists in gathering the application- and VM- level real time performance information, *i.e.*, the mean response time and the average memory utilisation, periodically using sensors. Then, during the analysis phase, the memory controller computes from the gathered information, which are used as decision making criteria, the amount of memory required by the application in order to meet its target performance. The strategy logic used in the

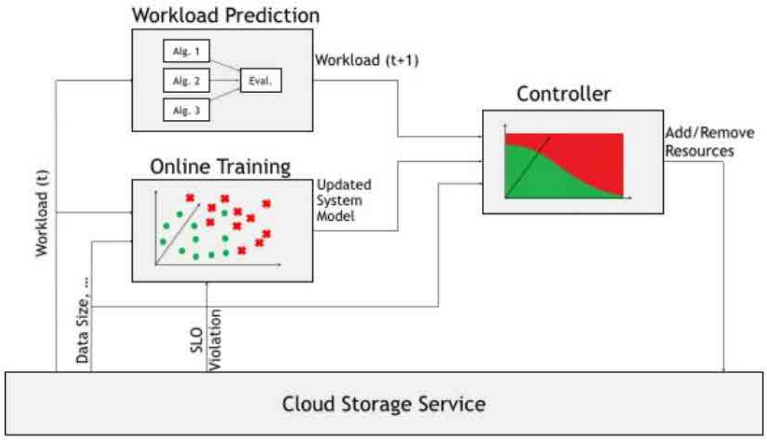


Figure 3.4: The architecture of OnlineElastMan [Liu 2016]

decision making is hard-coded in the controller. Therefore, during the final phases (*i.e.*, planning and execution), the actuator is invoked by the controller to either increase or decrease the allocated memory of the VM hosting the application at runtime according to the required memory previously computed in order to preserve the required level of the application performance, *i.e.*, response time.

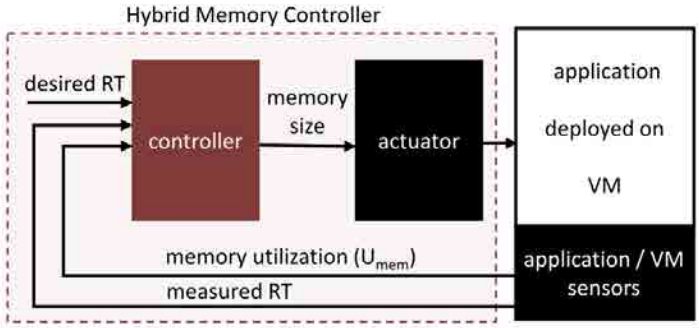


Figure 3.5: The architecture of the hybrid memory controller[Farokhi 2016]

Another work tackled the problem of memory elasticity of VMs has been proposed in [Moltó 2016]. The authors introduced a framework, named CloudVAMP (Cloud Virtual machine Automatic Memory Procurement), for monitoring VMs and adjusting their allocated memory to the memory requirements of their running applications using a cloud vertical elasticity controller/manager. This framework is proposed to be integrated with a Cloud Management Platform (CMP), to provide automatic vertical elasticity featuring live migration to prevent physical machine overloading. The ar-

chitecture of CloudVAMP as depicted in Figure 3.6 consists of three components: (1) a component named Cloud Vertical Elasticity Manager (CVEM) that runs alongside a CMP to obtain the monitoring information regarding the memory usage of all the VMs and analyse the amount of memory needed by the VMs according to an elasticity strategy, implemented in CloudVAMP, in order to dynamically update the memory allocated to each of them; (2) a component named Memory Reporter (MR) which is used to periodically report the free, used memory in the VMs to a monitoring system; and (3) a component named Memory Over-subscription Granter (MOG) which is used to inform the CMP about how much memory can be oversubscribed on the hosts to be considered by the scheduler of the CMP.

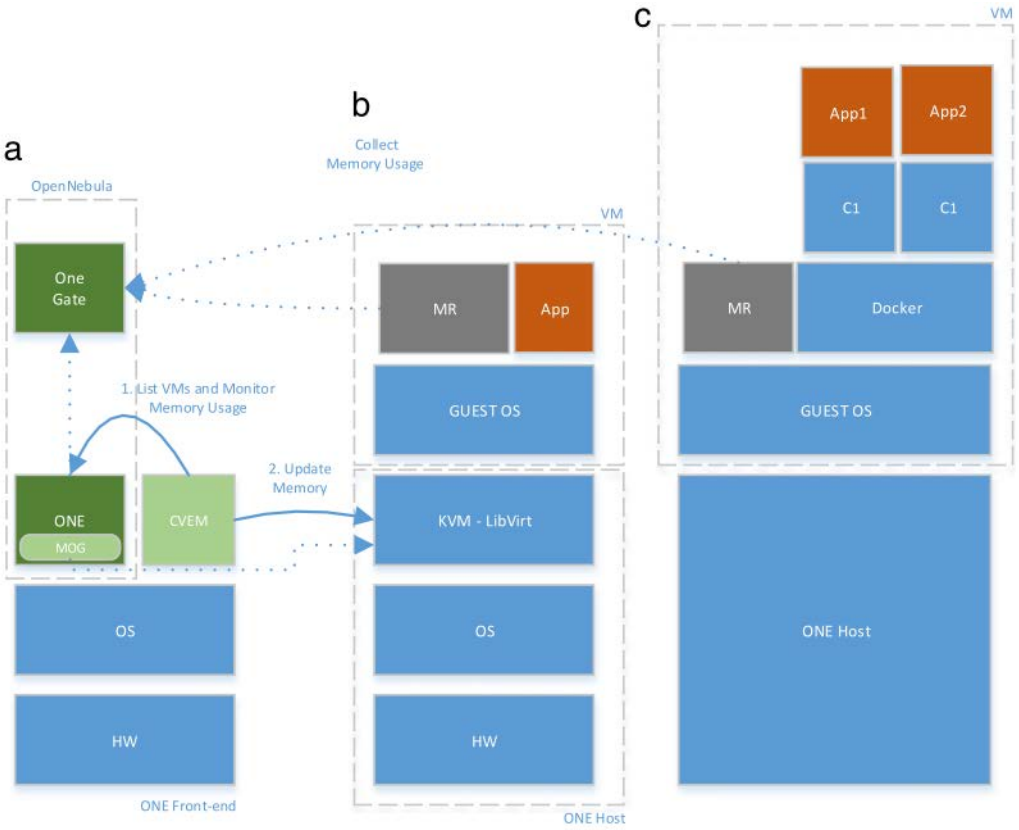


Figure 3.6: CloudVAMP architecture [Moltó 2016]

While most of the presented approaches are recently proposed, they focused on a specific elasticity model for constructing their elasticity controllers, which only tackle either horizontal or vertical elasticity at infrastructure scope.

3.3 Elasticity strategies description

In the last few years, researchers have started to pay attention to the importance of providing an unified way to define elasticity strategies that tend to become very complex and hard to develop in hard-coded way. We will present in the following the few attempts that have been made by the research community so far.

In [Copil 2013], a domain-specific language, called Simple Yet-Beautiful Language (SYBL), has been proposed for controlling elasticity in Cloud applications. SYBL is designed based on the use of programming directives (which are used to control work distribution and communication at runtime) to monitor and specify different constraints (describing QoS requirements) and strategies at different granularities, *i.e.*, application, component, and programming level. So, some elasticity specifications can be provided at the application level to describe certain global application characteristics such as application response time. Some others can be provided either at the component level to describe certain characteristics of a specific component, such as the average number of IOs of the component, or at the programming level to specify the elasticity requirements within the application component code, *e.g.*, data accuracy, CPU usage of a portion of code. Elasticity specifications are defined in SYBL using directives for monitoring, constraints and strategies, as shown in Grammar 3.1, and can be specified as Java annotations or as SYBL enriched XML descriptions. The monitoring directives are used to assign to a variable an existing cloud metric to be monitored or a formulas constructed from combining several cloud metrics. The constraints directives are used to allow to specify constraints established either on a simple metric or on a complex cloud metric determined by formulas. A constraint is used to describe the maximum and the minimum thresholds of the monitored cloud metrics. In order to describe the elasticity strategies, SYBL uses strategy directives that define elasticity strategies in the form of: (1) `Condition:Action` that triggers the execution of an elasticity action, *i.e.*, ScaleOut or ScaleIn, or a migration action, *i.e.*, Migrate, specified in case the condition is true, or (2) `WAIT Condition` that tells to wait until the condition became true. A condition in a strategy can be expressed as logical combination of constraints defined using constraint directives. The violation or fulfilment of those constraints can lead to triggering particular elasticity actions.

A DSL language, named Scalability Rule Language (SRL), has been proposed in [Kritikos 2014] for specifying scalability rules through defining event patterns of multi-Cloud application as well as scaling actions. SRL has been inspired by OWL-Q language for specifying QoS metrics as well as the Esper Processing Language (EPL) for specifying event patterns. Thus, it has been constructed as a modeling language that provides modeling concepts for defining scalability rules, events, conditions, and actions. Figure 3.7 illustrates SRL specification model for defining scalability rules, scaling policies, and actions. A scalability rule is associated with a set of actions, an event, a set of scaling policies that restrict how scaling actions are performed, and optionally entities representing the user or organisation owning the rule. An action in SRL can be either

$$M_i := \textbf{MONITORING}\ varName = x_j \mid \textbf{MONITORING}\ varName = formula(x_1 \dots x_n)\ where\ x_j \in,\ c \in ApplicationDescriptionInfo$$

$$C_i := \textbf{CONSTRAINT}\ p \in formula_i(x)\ rel\ formula_j(y)\ where\ x,\ y \in ApplicationDescriptionInfo,\ rel \in \{\leq,\ \geq,\ \neq,\ =\}$$

$$S_i := \textbf{STRATEGY CASE}\ [Condition : Action] \mid \textbf{WAIT}\ Condition \mid \textbf{STOP} \mid \textbf{RESUME} \mid \textbf{EXECUTE}\ strategyName\ parameter_{1\dots n}\ where\ Condition : DefFunctions \rightarrow \{true,\ false\}$$

Grammar 3.1: SYBL Language Constructs [Copil 2013]

a scaling action or an event creation action that represents the inability of scaling actions to maintain the required QoS for cloud application/components. The scaling action can be either a horizontal scaling action, *i.e.*, scale-out and scale-in actions to scale the number of VMs, or a vertical scaling action, *i.e.*, scale-up and scale-down actions, to scale VM's resources. The execution of these actions are triggered by the occurrence of an event. The latter is classified in SRL as a simple event or an event pattern representing a composition of events (simple events or other event patterns). A simple event can be either a functional event representing a functional error, or a non functional event representing the violation of a QoS metric that occurs when the current value of the QoS metric exceeds its defined threshold that is provided in SRL as a metric condition. All of these elements are the construction of a SRL conceptual model that represents scalability rules used to ensure application/component's elasticity. Though, the modeling aspect of SRL makes it effortful to use and express complex rules.

Contrary to our work, an elasticity rule in SYBL and SRL is associated to one specific component identified by its name. SBP holder cannot attach the same rule to different tasks/services. Moreover, these languages don't use symbolic constants and embed rather constant values directly in rule specifications what makes rule definitions and maintenance difficult.

Another work has been presented in [Zabolotnyi 2015] where the authors proposed

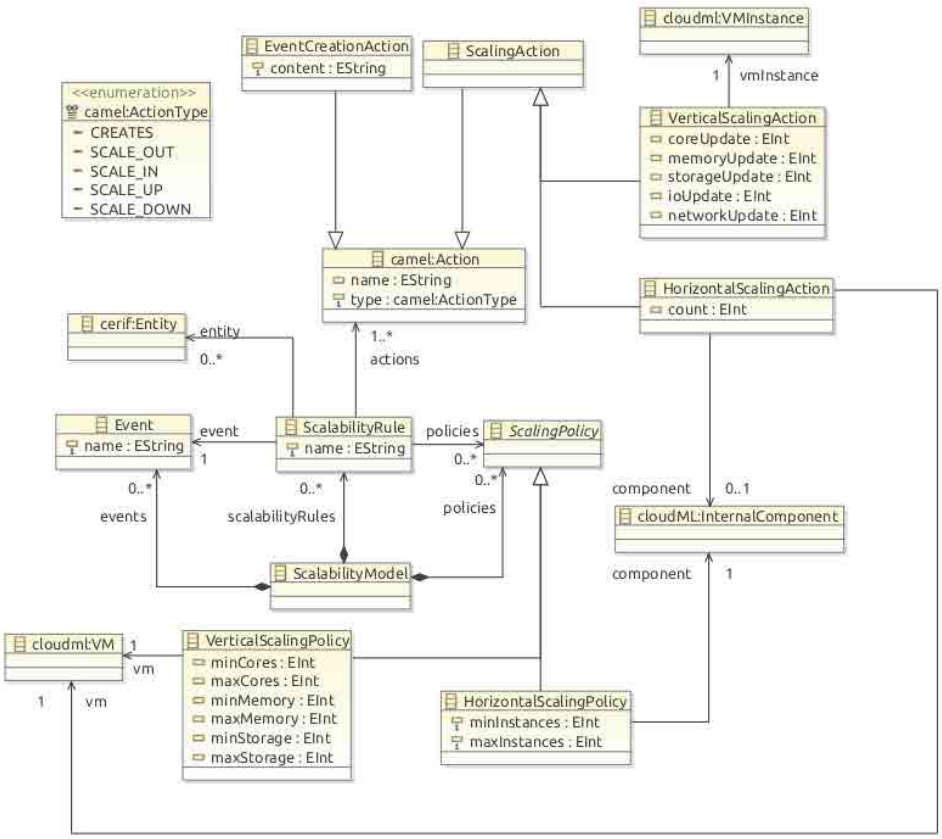


Figure 3.7: SRL Modeling Constructs - Scalability Rules Specification [Kritikos 2014]

a declarative domain-specific language, called SPEEDL, for specifying scaling policies for applications deployed in IaaS clouds. SPEEDL allows to define scaling policies as a set of event-condition-action (ECA) rules for resource management as well as task mapping. A scaling policy in SPEEDL can be seen as a 2-tuple $TP =< TM, RM >$ where TM denotes a set of task management rules and RM denotes a set of resource management rules. The top level of SPEEDL grammar for defining scaling policy is given in Grammar 3.2. The user is allowed using SPEEDL to describe a scaling policy by providing a sequence of task management and/or resource management rules followed by optionally a validation statement that allows to check the consistency of rules, *i.e.*, internal rule validation, and a terminal statement. Rules are of four types: scheduling and migration rules for task management, and scale-up and scale-down rules for resource management. Resource management rules are used to manage the adding and removing of VMs to/from the managed application according to the workload change to meet the application QoS requirements. This happens through performing scale-up action for adding VMs and scale-down for releasing them. On the other hand, task management

rules are used in SPEEDL to map tasks to resources through either task scheduling or task migration. Task scheduling represents the mapping of a new task to a resource while task migration consists on arranging and moving tasks from one resource to another in order to maintain overall system stability.

⟨*ScalingPolicy*⟩ ::= ⟨*SPConigElements*⟩

⟨*SPConfigElements*⟩ ::= ⟨*Rule*⟩ ⟨*SPConfigElements*⟩
| ⟨*Validation*⟩ ⟨*SPTerminalStatement*⟩
| ⟨*SPTerminalStatement*⟩

⟨*SPTerminalStatement*⟩ ::= '`build`'

⟨*Rule*⟩ ::= ⟨*ScaleUpRule*⟩
| ⟨*ScaleDownRule*⟩
| ⟨*SchedulingRule*⟩
| ⟨*MigrationRule*⟩

Grammar 3.2: General SPEEDL Language Specification [Zabolotnyi 2015]

A part from language related aspects; these works don't take into consideration fundamental characteristics of service-based processes, which make them unsuitable for defining elasticity strategies for elastic SBPs. Indeed, the executions of process instances are scattered over a set of services related to each other according to the process control flow. First, these services may have different resource requirements and have thereafter different elastic behavior. Second, due to task/service dependencies prescribed by the control flow an elasticity strategy of a given service may need to refer to other related services' states. It is not clear how current approaches can expand their local analysis of the monitored information to have a more global view.

3.4 Elasticity strategies evaluation

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

In [Copil 2015], Copil *et al.* have proposed a framework, named ADVISE (evAluating clouD serVIce elaSticity bEhavior), for the evaluation of Cloud service elasticity behavior. It has been designed based on a learning process and a clustering-based evaluation process that allows to determine at runtime the expected elasticity behavior of Cloud service. ADVISE can be used to improve the quality of elasticity controller decisions as well as to evaluate different elasticity control processes (*i.e.*, elasticity enforcement plans composed actions to be applied to enforce a service/component elastic-

ity) and determine the most appropriate one for the service/component in a particular situation. As shown in Figure 3.8, the learning process receives periodically current information about the metrics that might have influence on the Cloud service behavior such as service structure and workload, deployment strategies, the control processes enforced and the resource used by the service, *etc.* Then, it transforms them to multi-dimensional points to detect and evaluate the expected elasticity behavior. ADVICE is designed to be integrated with elasticity controllers allowing evaluating at runtime elasticity control processes for their elasticity capabilities.

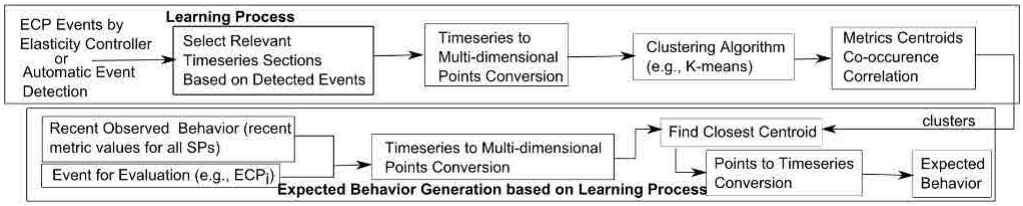


Figure 3.8: Modeling cloud service behavior process [Copil 2015]

In [Naskos 2015b], Naskos *et al.* proposed a formal model for quantitative analysis of horizontal elasticity at the infrastructure scope using Markov Decision Processes (MDPs). The model has been defined to formally control at runtime the adding and removing of VMs to/from the managed system. Figure 3.9 shows a MDP model that is represented as a set of states and the enabled elasticity actions in each state. A state in the model corresponds to the number of VMs provisioned for the application. A transition is connecting a source state to a target state which corresponds to the execution of an enabled action (*i.e.*, adding/removing of VMs or doing nothing). The transitions in the model are mapped to a probability representing the possibility of meeting the response latency threshold when applying a specific action based on the future incoming load and the collected logs. Based on MDP models, the authors have proposed continuous online verification in order to execute specific elasticity actions. This is done by dynamically instantiate MDPs at runtime according to the current workload and environment conditions and verifying them using a model checker for provisioning/deprovisioning of cloud resources. This approach allows the runtime evaluation of different elasticity strategies for horizontal scaling in terms of maximizing the system utility (*i.e.*, avoiding over-provisioning).

However, despite the usefulness of runtime approaches in improving the used elasticity strategies, the effectiveness of the latter have to be guaranteed before putting them in use in real cloud environments to prevent any suspicious elasticity behavior that can be detected during the strategies design time.

Suleiman *et al.* have proposed in [Suleiman 2013] an analytical model, using queuing theory, to study the effects of elasticity strategies on the performance of multi-tier applications (more particularly three-tier applications) deployed on cloud infrastructures.

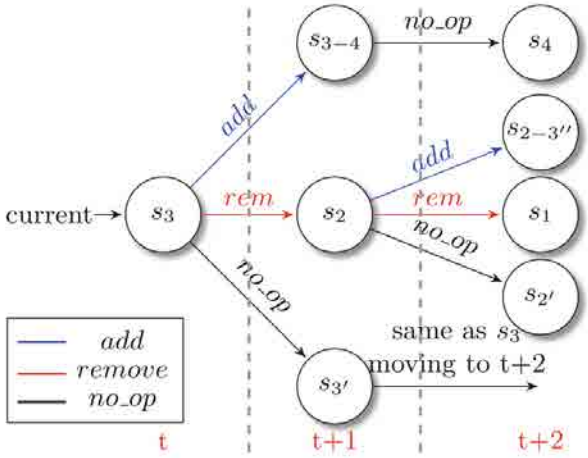


Figure 3.9: Markov Decision Process (MDP) model overview [Naskos 2015b]

Particularly, they studied the impact of changing the threshold values, used by the elasticity strategy, on application performance. The proposed model allows application owners to evaluate different thresholds-based elasticity strategies and choose the most effective one for their applications. The authors claim that their model enables the application owners to determine approximately the values of CPU utilization, response time of the application, and the number of servers needed to handle their application workload. Figure 3.10 illustrates the authors' model for three-tier applications architecture using queuing theory. Each tier is assumed to be deployed on separate servers on a public cloud infrastructure. Each server in a tier is represented as a queue. The first tier, *i.e.*, the web server, acts as a load balancer which balances the incoming requests between application servers in the second tier. To serve a request, an application server sends a set of queries to the database server. The authors focused in their work on modeling the elasticity of the application tier. So, they modelled the application tier as an M/M/m queue where m is a variable representing the number of servers and the requests arrival is represented as a Poisson process. The model has a set of parameters defined for a particular time interval. These parameters are used to approximate the average CPU utilization and the mean application's response time during a time interval. Based on their model, the authors proposed two elasticity algorithms that simulate scale-out and scale-in logic for horizontal scaling based on CPU utilization. The scale-out and scale-in algorithms compare the value of the estimated CPU utilization to the given CPU utilization threshold in order to decide to perform elasticity actions on the cloud infrastructures.

In [Amziani 2015],the author proposed an approach for ensuring and evaluating SBPs elasticity in the cloud by defining: (i) a formal model representing the deployment model of SBPs associated with two elasticity mechanisms (Duplication/Consolidation)

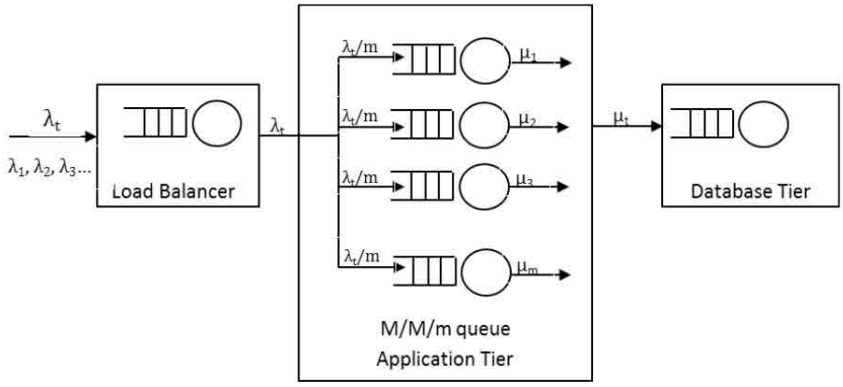


Figure 3.10: The analytical model for elasticity evaluation [Suleiman 2013]

for horizontal scaling that allow to add/remove service's copies in order to preserve some QoS requirements (process/service level) and (*ii*) an elasticity controller that is used to ensure SBPs elasticity through applying the defined elasticity mechanisms according to a given elasticity strategy and allows to evaluate different elasticity strategies. The formal model for SBP elasticity has been modelled based on Petri nets to describe their composed services and the relationship between them without considering the distinguish characteristics of services requests. Using this formal model, the authors have defined and formalized the elasticity operations representing the duplication and consolidation mechanisms. The duplication operation is defined to ensure that only overloaded services are duplicated while the consolidation operation is defined to prevent the over-provisioning of services. These operations are the basis for constructing the elasticity controller which is modelled as a high-level petri net as shown in Figure 3.11.

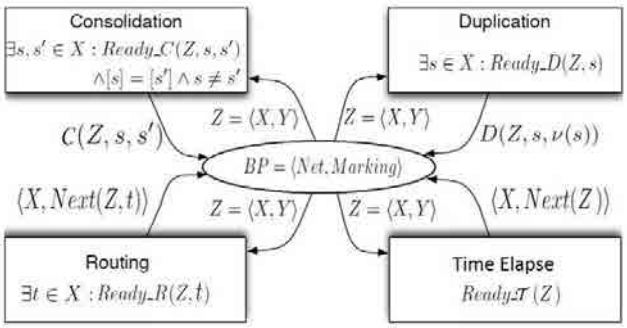


Figure 3.11: High-level petri net model of the elasticity controller for evaluating SBPs elasticity [Amziani 2015]

The high-level petri net model of the controller contains one place of type net system

representing the SBP petri net model. It is composed of four transitions representing the main actions, *i.e.*, calls transfer (Router), the elapse of time (Time elapse), the duplication of an overloaded service (Duplication) and the consolidation of an underused service (Consolidation), that the controller could perform on the SBP petri net model and makes it evolve over time. Each transition is guarded by a condition that decides when, where and how to perform the action. The provided conditions and their associated actions represent a specific elasticity strategy. Based on its formal aspect, the controller is able to evaluate the effectiveness of different elasticity strategies, described for horizontal scaling, before using them in the cloud.

In [Evangelidis 2018], Evangelidis *et al.* proposed an approach based on performance modeling and formal verification to provide performance guarantees on auto-scaling strategies that describe rules for scaling-out and scaling-in cloud-based applications. The application performance is modelled as a probabilistic model using a discrete-time Markov chain (DTCM) in which the model's states represent the performance information (*i.e.*, CPU utilisation and/or response time) required to capture the impact of the auto-scaling strategy on QoS (*i.e.*, CPU utilisation and/or response time) while the transitions between states capture the probability of changing from a state to another particular state when applying scale-out or scale-in actions. The authors have proposed to used a clustering algorithm, *i.e.*, k-means, to initialize their model from the CPU utilisation and response time traces gathered from running the application on different number of VMs rented from either Amazon EC2 or Microsoft Azure. The number of generated clusters for a single run of k-means algorithm is considered as the number of possible states to which the application can go when a scaling action occurs. Based on the probabilistic model, the verification process start by passing an auto-scaling strategy or a set of strategies to verify some defined properties such as checking whether the cloud application will end up in a state where the QoS requirements (*i.e.*, maximum CPU utilisation and maximum response time) will be violated. Despite the importance of providing formal verification to guarantee QoS requirements with respect to a given strategy, the proposed verification approach is not suitable and could not be applied for SBPs in which QoS requirements could be provided for different levels (*i.e.*, process, service and request level). To the best of our knowledge, existing model checkers are not applicable for formally verifying the elasticity of SBPs at service level considering QoS requirements for requests. Moreover, the proposed approach requires application owners to run their target application on a real cloud environment in order to be able to evaluate the defined auto-scaling strategies.

3.5 Synthesis of Related Work

In this chapter, we proposed an overview on existing works that aim at (*i*) proposing models and mechanisms for ensuring elasticity, (*ii*) describing elasticity strategies and (*iii*) evaluating them. In this section, we synthesize the studied research works, as illustrated in Table 3.1, based on their general characteristics and how they respond

to the challenges presented in Chapter 1. For each work, we highlight the following characteristics :

1. ***Target :*** represents the kind of applications for which the proposed solution is designed, *e.g.*, a simple cloud application/service, a SBP *etc.*;
2. ***Phase :*** indicates in which phase of system life-cycle the proposed solution can be used. We distinguish two main phases : (*i*) design phase and (*ii*) execution phase (equivalent to enactment phase in business process life-cycle);
3. ***QoS requirement level :*** QoS requirements can be specified for the solution's target at different granularity levels. This characteristic indicates the different levels at which the solution allows QoS requirements to be specified;
4. ***Elasticity Capability :*** different elasticity capabilities can be considered by the solution: "Horizontal", "Vertical", "Hybrid" or "Manual";
5. ***Level of decoupling :*** indicates whether the solution decouples the elasticity information from the elasticity controller allowing different elasticity information to be considered (provided) and at which level the decoupling is made. We distinguish two levels of decoupling : (*i*) decoupling elasticity strategies from elasticity controller (ES/EC), (*ii*) decoupling between elasticity strategies, elasticity model and elasticity controller (ES/EM/EC). The value '-' indicates the absence of decoupling which means that elasticity information is hard-coded in the controller, if exists;
6. ***Support :*** indicates whether a comprehensive support system is provided by the solution.

Studied solutions	*Target*	*Phase*	*QoS requirement level*	*Elasticity Capability*	*Level of decoupling*	*Support*
Models and Mechanisms of elasticity						
[Bessai 2012b, Bessai 2012a]	Business process	Execution	Process	Manual	-	-
[Hoenisch 2014]	SBP	Execution	Process/ Service	Horizontal	-	-
[Ali-Eldin 2012]	Cloud service	Execution	Service	Horizontal		
[Loff 2014]	Cloud application	Execution	Application	Horizontal	ES/EC	-
[Liu 2016]	Storage system	Execution	Network	Horizontal	-	-
[Farokhi 2016]	Cloud application	Execution	Application	Vertical	-	-
[Moltó 2016]	Cloud application	Execution	Application	Vertical	-	-
Elasticity strategies description						
[Copil 2013]	Cloud application	Design	Application/ Component/ Programming	Horizontal	ES/EC	-
[Kritikos 2014]	Cloud application	Design	Application/ Component	Horizontal Vertical	ES/EC	√
[Zabolotnyi 2015]	Cloud application	Design	Application	Horizontal	ES/EC	√
Elasticity strategies Evaluation						
[Copil 2015]	Cloud service	Execution	-	Controller's capabilities	-	-
[Naskos 2015b]	Cloud system	Execution	-	Horizontal	-	-
[Suleiman 2013]	Cloud application	Design	Application	Horizontal	-	-
[Amziani 2015]	SBP	Design	Service	Horizontal	ES/EC	-
[Evangelidis 2018]	Cloud application	Design	Application	Horizontal	ES/EC	-

Table 3.1: Synthesis of Related Work.

In the light of the aforementioned works and their synthesis in Table 3.1, it is noticeable that even with the remarkable literature works and efforts that have been made during the recent years in order to provide efficient elastic systems, there is still lack of supports that can be characterized by being comprehensive, granular (in terms of supporting QoS requirements) and generic, especially for SBPs. The related work limitations regarding our objective can be summarized as follows :

- ***Granularity :*** Even though some of the studied approaches [Hoenisch 2014, Copil 2013, Kritikos 2014] have tried specifying QoS requirements (*e.g*, the defined thresholds for QoS metrics) at different granularity levels, none of them have considered the request level, *i.e.*, distinguishing between requests requirements. In fact, all of them have assumed that QoS related requirements are the same for all requests. However, enactment requests of a SBP are different and require therefore different amount of resources. Especially when dealing with data-aware SBPs, some requests can be more data-intense than others, which could make a huge influence on the QoS if we handle them in the same manner.

- ***Genericity :*** Except the work of [Copil 2015] that detects the elasticity capabilities of the control system (*e.g.*, elasticity controller) at runtime to evaluate the effect of the used elasticity strategy during the application execution in the cloud, all of the proposed approaches have been designed according to a specific elasticity model restricting the definition and use of elasticity strategies to a particular constant set of actions. Though some authors have proposed to decouple elasticity strategies from the controller in order to generalize the use of their systems, they are still tied to a specific elasticity model, therefore to particular elasticity capabilities and QoS metrics. None of them have considered the decoupling between elasticity strategies, elasticity model and elasticity controller (ES/EM/EC).

- ***Comprehensive :*** Our last findings concerns the absence of user-friendly supports for defining and evaluating elasticity strategies. Some authors [Kritikos 2014, Zabolotnyi 2015] have indicated providing users with a support to facilitate the definition of their elasticity strategies using the proposed DSL. However, none of the studied solutions for evaluating elasticity strategies have been designed with a way to facilitate their use by IT manager or cloud system designer.

3.6 Conclusion

In this chapter, we presented an overview of the existing works on different aspects related to elasticity models/mechanisms and elasticity strategies. It is worthy to say that some works could have been presented for more than one area such as the work of [Amziani 2015] which provided an attempt for elasticity strategy evaluation and proposing elasticity mechanisms for horizontal scaling. We started this chapter by discussing recently proposed works that propose models and mechanisms for ensuring systems

elasticity. Then, we studied the languages that have been proposed to facilitate the definition of elasticity strategies. Thereafter, we investigated how researchers have tackled the problem of evaluating elasticity strategies. Table 3.1 presents an illustrative picture to show the limitations of the studied works regarding our research objective. We start presenting in detail our contributions in the next chapters. In Chapter 4, we will propose a formal elasticity approach for data-aware SBPs allowing the evaluation of elasticity strategies before using them in the cloud. The approach is composed of a formal model for describing elastic execution environments for SBPs while considering the distinguish requirements for services requests, and elasticity mechanisms/capabilities for hybrid scaling. In Chapter 5, we will present two domain-specific languages for describing elasticity models and elasticity strategies. Thereafter, we will propose an end-to-end framework for evaluating elasticity strategies in Chapter 6.

CHAPTER 4

Modeling of Data-aware Elastic SBPs

Contents

4.1 Introduction

In this chapter, we propose a data-aware formal model that describes the elastic execution environment of SBPs. The execution environment of a given elastic SBP can be seen as a network, of service engines (refer to containers such as Docker [Merkel 2014], micro-container [Yangui 2011]) isomorphic to the SBP model. The SBP instance (request) execution is routed over several services (execution) engines that match each of the SBP component services via routers which represent resources that provide message format transformations and routing. The execution environment structure of a SBP evolves over time according to requests load by adding/removing service engine

copies to meet its QoS requirements (*e.g.*, maximum response time). The copies of a service engine are related to a load balancer service to balance the incoming load between them. In our data-aware formal model, we describe the characteristics of services engines composing an execution environment of a SBP and their requests. Such a model allows users distinguishing between different requests requirements when defining elasticity strategies. Furthermore, our model formalizes elasticity mechanisms for hybrid scaling which allows adding/removing service engine copies with different configurations. The use of hybrid scaling method allows to customize the provided resources according to service's requests characteristics. Applying the defined elasticity mechanisms on the formal model according to a specific elasticity strategy changes its structure and status allowing the evaluation of the elasticity strategy and ensuring its effectiveness before using it in real Cloud environment.

This chapter is organized as follows: Section 4.2 presents some basic mathematical notations and petri net related concepts used in the remainder of this chapter. In Section 4.3, we introduce our approach for data-aware modeling of elastic SBPs. Then, we propose elasticity mechanisms for ensuring hybrid elasticity of SBPs at service level (*cf.*, Section 4.4).

4.2 Preliminaries

Before presenting our formal model, we present a simplified definition for some used notions throughout in this chapter, namely sets, multi-sets, and high-level petri nets.

4.2.1 Basic notions

In this section, we introduce the basic notations, for sets and multi-sets. A set is defined as follows:

Definition 1 (Set) *A set S is an infinite collection of elements listed between braces, e.g. , $S = \{a, b, c, d\}$. The empty set is denoted by $\emptyset$. $\|S\|$ denotes the set's size, i.e., the number of elements in the set S. For example, $\|S\| = 4$. P(S) denotes the power set of S which is the set of all subsets of S, including the empty set and S itself. For example P(S) = $\{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b, c, d\}\}$.*

Three operations are defined over sets, namely, union, intersection and difference. Let $S_1 = \{a, b, c\}$ and $S_2 = \{b, c, d\}$ be two sets. The union of S_1 and S_2 is a set of elements which are in S_1, in S_2, or both. $S_1 \cup S_2$ denotes the union of the two sets. For example $S_1 \cup S_2 = \{a, b, c, d\}$. The intersection of S_1 and S_2 is a set of elements which are in both S_1 and S_2. $S_1 \cap S_2$ denotes the intersection of the two sets. For example, $S_1 \cap S_2 = \{b, c\}$. The difference between S_1 and S_2 is a set of elements which are in S_1 and not in S_2. $S_1 \setminus S_2$ denotes the difference between the two sets. For example $S_1 \setminus S_2 = \{a\}$.

A multi-set is a generalization of a set. We define it as follows:

Definition 2 (Multi-set) *A multi-set m over a set S is a function that maps each element $s \in S$ into a non-negative integer $\mathbb{N}$ representing the number of appearances (coefficient) of s in the multi-set M. We denote by m(s) the number of appearances of s in m. The element s is a member of a multi-set m iff $\forall s \in S : s \in m \Leftrightarrow m(s) > 0$. S_{MS} denotes the set of all multi-sets over S. The empty multi-set is denoted by $\emptyset_S$. A multi-set m is usually represented by the formal sum notation as follows:*

$$m = \sum_{s \in S} m(s)\,'s$$

An example of a multi-set m over $S = \{a, b\}$ is m = (2'a + 1'b).

The intersection and union operations defined over sets can be extended to multi-sets. Let's m_1 = (1'a + 1'b) and m_2 = (2'a + 1'c) be two multi-sets over S = $\{a, b, c\}$. The intersection of m_1 and m_2 is a multi-set m_3 where for all s $\in S$: $m_3(s) = \min(m_1(s), m_2(s))$. For example (1'a + 1'b) $\cap$ (2'a + 1'c) = (1'a). The union of m_1 and m_2 is a multi-set m_3 where for all s $\in S$: $m_3(s) = \max(m_1(s), m_2(s))$. For example (1'a + 1'b) $\cup$ (2'a + 1'c) = (2'a + 1'b + 1'c).

4.2.2 Petri nets

Petri nets [Murata 1989] are formal models used to describe concurrent distributed systems. They are divided into low-level Petri Nets and high-level Petri Nets. Low-level Petri Nets, simply called Petri Nets or Place/Transition nets, have been widely used to model, analyse and verify business processes [van der Aalst 1998, van der Aalst 2011] for their mathematical foundation which allows to apply various analysis techniques. A Petri net is considered as a bipartite directed graphs with four types of objects, *i.e.*, places (represented with circles), transitions (represented with rectangles), directed arcs connecting either places to transitions or transitions to places, and tokens (represented with black dots) (*cf.*, Figure 4.1(a)). A Petri net is defined as follows.

Definition 3 *A petri net is defined as 5-tuple S_N = (P, T, Pre, Post, M_0) where:*

- *$P = \{P_1, P_2, \ldots, P_n\}$ is a set of places;*
- *$T = \{T_1, T_2, \ldots, T_m\}$ is a set of transitions;*
- *Pre: $P \times T \rightarrow \mathbb{N}$ is a function indicating the input arcs from places to transitions;*
- *Post: $T \times P \rightarrow \mathbb{N}$ is a function indicating the output arcs from transitions to places;*
- *M_0 is an initial marking.*

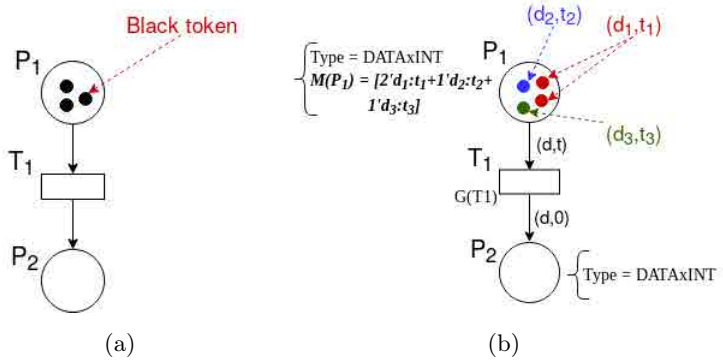


Figure 4.1: Simple examples of (a) a classic petri net and (b) a high-level petri net

A petri net structure is denoted by N = (P, T, Pre, Post) without the initial marking. The petri net system can be denoted by S_N *= (N,* M_0*).*

High-Level Petri Nets have been proposed to extend low-level petri nets by introducing higher-level concepts, such as representing tokens by complex data structure, and using expressions to annotate net elements (*i.e.*, places, transitions, and arcs) [Jensen 2009]. They have been widely used for modeling and simulating engineering and scientific problems with complex structures. Colored Petri nets and Timed Coloured Petri nets represent popular classes of high-level Petri nets. Their main advantage over other Petri nets classes is that tokens involved in Timed/Untimed Colored Petri nets are distinguishable by their attached data values called colors, so, each token has a set of attributes characterising it. In addition to data values, tokens in Timed Colored Petri nets have also timing information attached to them allowing to model and validate real-time systems. So, each token can carry a second value along with its color, called a timestamp. Figure 4.1(b) illustrates an example of a Timed-Colored Petri net composed of a transition and two places of type DATA×INT. One of the places contains four tokens representing its marking where two of token have a data value $d_1 \in$ DATA and a timestamp t_1. A marking of a place in Timed-Colored Petri nets is a timed multi-set. A timed multi-set has been proposed as an extension for a multi-set.

A timed multi-set tm over a non-empty set S specifies the elements in the multi-set over S together with their number of appearances and their timestamps. An element in a timed multi-set tm is often represented as $tm(s,t)'s : t$ where s is an element of S, t is a timestamps and $tm(s,t)$ is the number of appearances of s with the timesptamps t. So, the timed multi-set tm can be viewed as a function that determines the number of appearances for each pair (s,t) and it is represented as a summation of its elements $tm(s,t)'s : t$. S_{TMS} denotes the set of all timed multi-sets over S.

Definition 4 (Timed Multi-set) *Let S be a non-empty set. A timed multi-set tm over S is a function that maps each element (s,t) from* $S \times \mathbb{N}$ *into a non-negative integer* $\mathbb{N}$ *representing the number of appearances (coefficient) of (s,t) in the timed multi-set tm.*

The number of appearances tm(s,.) of an element $s \in S$ is the number of times that s appears with some timestamp t in tm, i.e., $tm(s,.) = \sum_{t \in \mathbb{T}} tm(s,t)$. The element s is a member of a timed multi-set tm iff $\forall s \in S : s \in tm \Leftrightarrow tm(s,.) > 0$. S_{TMS} denotes the set of all timed multi-sets over S. The empty timed multi-set is denoted by $\emptyset_S$. We often represent the timed multi-set tm using the formal sum notation as follows:

$$tm = \sum_{s \in S} \sum_{t \in \mathbb{T}} tm(s,t)\ 's : t$$

Example 1 *Let's consider a color set $C = \{a, b, c, \ldots\}$, the timed multi-sets (3'a:3), (1'b:3 + 1'd:5), (1'a:0 + 2'b:5 + 3'c:3 + 4'd:10) and $\emptyset_C$ over C are members of C_{TMS}.*

Definition 5 (Timed Multi-set addition) *Let tm_1 and tm_2 be two timed multi-sets over a set S. The addition of tm_1 and tm_2, noted $tm_1 +\!+\!+ tm_2$, is computed by adding for each element (s,t) its coefficient $tm_1(s,t)$ in tm_1 and its coefficient $tm_2(s,t)$ in tm_2, i.e.,*

$$tm_1 +\!+\!+ tm_2 = \sum_{(s,t) \in tm_1 \cup tm_2} (tm_1(s,t) + tm_2(s,t))\ 's{:}t$$

Definition 6 (Timed Multi-set inclusion) *Let tm_1 and tm_2 be two timed multi-sets over a set S. The tm_1 is a sub-timed multi-set of or equal to tm_2, noted $tm_1 \subseteqq tm_2$, iff each element (s,t) of tm_1 is also a member of tm_2 and its coefficient $tm_1(s,t)$ in tm_1 is less than or equal to its coefficient $tm_2(s,t)$ in tm_2, i.e., $\forall$ (s,t) $\in tm_1$, $tm_1(s,t) \leq tm_2(s,t)$.*

Definition 7 (Timed Multi-set comparison) *Let tm_1 and tm_2 be two timed multi-sets over a set S. The tm_1 is smaller than or equal to tm_2, noted $tm_1 \lll= tm_2$, iff* (1) *the number of appearances of each element s in tm_1 is less than or equal to the number of appearances of the element s in tm_2, i.e., $\forall\ s \in tm_1$: $tm_1(s,.) \leq tm_2(s,.)$, and* (2) *the largest timestamp t_1 of the element s in tm_1 is smaller than or equal to the largest timestamp t_2 of the element s in tm_2, i.e., $\forall$ (s,t_1) $\in tm_1$ $\exists$ (s,t_2) $\in tm_2$: $t_1 \leq t_2$.*

Definition 8 (Timed Multi-set subtraction) *Let tm_1 and tm_2 be two timed multi-sets over a set S. The subtraction of tm_1 from tm_2, noted $tm_2 - - - tm_1$, is computed when $tm_1 \lll= tm_2$ by removing $tm_1(s,.)$ occurrences of the element s in tm_1 from the occurrences of s in tm_2 that have the largest timestamps, i.e., the oldest elements (s,t) in tm_2 are removed first for each element s in tm_1.*

A formal definition of Timed-Colored Petri nets is given as follows. $EXPR$ is used to define an expression which can be a function, a variable, a color constant, or an empty expression denoted by $\emptyset$.

Definition 9 (Timed-Colored Petri net) *A Timed-Colored Petri net is formally defined as a nine-tuple S_{TCN}= (Σ, P, T, σ, V, $\mathcal{G}$, Pre, Post, M_0) where:*

- Σ *a set of types. Each type corresponds to a set of colors.*
- $P = \{P_1, P_2, \ldots, P_n\}$ *is a set of places;*
- $T = \{T_1, T_2, \ldots, T_m\}$ *is a set of transitions;*
- σ*:* $P \rightarrow \Sigma$ *is a function that assigns a color set to each place in* P*.*
- V*: a set of typed variables such that for all* $v \in V$ *:* $Type(v) \in \Sigma$ *.*
- $\mathcal{G}$*:* $T \rightarrow EXPR$ *is a guard function that assigns a boolean expression to each transition* T_i *to control the flow of tokens passed by the transition.*
- *Pre:* $P \times T \rightarrow EXPR$ *in an input arc expression. An empty expression* $\emptyset$ *indicates the absence of an arc from a place* P_i *to a transition* T_i*.*
- *Post:* $T \times P \rightarrow EXPR$ *is an output arc expression. An empty expression* $\emptyset$ *indicates the absence of an arc from a transition* T_i *to a place* P_i*.*
- M_0 *is an initial marking that maps each place* $P_i \in P$ *into a timed multi-set of tokens in* $\sigma(P_i)_{TMS} \rightarrow \Sigma_{TMS}$*.*

A Timed-Colored Petri net structure is denoted by TCN = (Σ*, P, T,* σ*, V,* $\mathcal{G}$*, Pre, Post) without the initial marking. The Timed-Colored Petri net system can be denoted by* S_{TCN} *= (TCN,* M_0*).*

4.3 Formal Modeling of data-based SBP

In the following sections, we present our formal model for describing, the structure and the behavior of elastic execution environments for SBPs.

4.3.1 Structural modeling

Our model is based on High-Level Petri Nets [Jensen 1991] to describe the service engines, hosting services in a SBP, and their requests, by allowing distinguishing between them, which makes it possible to define more sophisticated elasticity strategies using different elasticity indicators. The high-level concepts introducted in high-level petri net make the description of the elements of our execution environment of SBPs possible. Thus, our model, named *Time Colored-Service based Process (TC-SBP) petri net model*, is composed of a place denoting either a service engine or a load balancer, a transition denoting a router and a token denoting a service request/instance which may carry a data value represented by a token color.

A service (engine) place is characterised by a capacity indicating how much data the service engine can process simultaneously, a processing speed, and a function denoting the time complexity of the corresponding hosted service which allows to estimate the

processing time required by the service to handle a given request. Also, it can be specified either as an elastic service engine or not. So, if the service engine is considered as an elastic one, it is allowed to have many copies of it hosting the same service related to each other by an equivalence relation and connected to a load balancer. A load balancer place, *aka 'buffer place'*, is characterized by a capacity property representing its queue length. In addition, a temporal information can be provided to a buffer place as well as a service engine place to specify when a request should become outdated. A service request which is represented as a token in our model is characterized by a data size, a belonging category which might represents a tenant or defined according to the data size, and a state indicating the request progress inside the service engine place which can be either in waiting state, under-processing or finished. These properties represent the request/token's data value. A request might have also a processing/waiting time representing its age in a place. A data value with an age for a request/token is called timed token color. Each place in our model can hold many tokens representing its marking. So, the tokens distribution in our TC-SBP Petri net is called TC-SBP marking (*cf.*, Section 4.3.2)). The marking is dynamic and changes over time when requests/tokens move from service to service in the SBP until the end of the process. A transition is used as a router to connect places in our model to allow transferring tokens between them according to the behavioral specification of the SBP. Finally, places and transitions are connected through arcs that allow to modify the characteristics of tokens when they are "transferred" from one place to another using expressions on them.

In the following, we introduce the definition of our TC-SBP petri net model. Given an expression $e \in EXPR$, Var(e) is used to indicate the set of variables appearing in e.

Definition 10 (TC-SBP Petri Net model) *A Time Colored-Service based Process (TC-SBP) petri net model is a Petri net* $N = < \Sigma, P, T, \sigma, Capacity, Speed, Elastic, V, \mathcal{C}, \mathcal{G}, Pre, Post, \equiv_P, \equiv_T, \mathcal{I} >$ *where:*

- Σ*: a set of types. Each type corresponds to a set of colors which may be attached to one of places.*
- $P = S \cup B$ *a set of places where* S *and* B *represent respectively a set of service (engine) places and a set of buffer places corresponding to load balancer services.*
- T*: a set of transitions connecting places.*
- σ*:* $P \rightarrow \Sigma$ *is a function that assigns a color set to each place in* P*.*
- *Capacity:* $P \rightarrow \mathbb{N}$ *is a function that assigns a natural integer to each place representing its capacity.*
- *Speed:* $S \rightarrow \mathbb{N}$ *is a function that assigns a natural integer to each service place representing the number of instructions that can be processed per time unit.*
- *Elastic:* $S \rightarrow BOOLEAN$ *is a function that indicates if a service (engine) place is an elastic one or not.*

- *V: a set of typed variables such that forall* $v \in V$ *:* $Type(v) \in \Sigma$ *.*
- $\mathcal{C}$*:* $S \rightarrow EXPR$ *is a complexity function that assigns an expression to each service place. The expression is used to estimate the execution time of a request in a service engine.*
- $\mathcal{G}$*:* $T \rightarrow EXPR$ *is a guard function that assigns a boolean expression to each transition tr to control the flow of tokens passed by the transition.*
- *Pre:* $P \times T \rightarrow EXPR$ *in an input arc expression. An empty expression* $\emptyset$ *indicates the absence of a connection between a place p and a transition t through an input arc.*
- *Post:* $T \times P \rightarrow EXPR$ *is an output arc expression. An empty expression* $\emptyset$ *indicates the absence of a connection between a place p and a transition t through an output arc.*
- $\equiv_P \subseteq P \times P$*: represents an equivalence relation over P that identifies the set of service's copies.*
- $\equiv_T \subseteq T \times T$*: represents an equivalence relation over T that identifies the set of transition's copies.*
- $\mathcal{I}$*:* $P \rightarrow I$ *is an assignment of timing requirement to each place in P to indicate the staying time interval for a token. A token that stays in a place from more than the upper bound of its time interval is considered as outdated. I represents the set of time intervals defined by* $I ::= [a,a]|[a,b]|[a,\infty[$ *where* $a, b \in \mathbb{N}$ *and* $a < b$*.*

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

- $[p]_{\equiv_P} = \{$p' | (p,p') $\in \equiv_P\}$
- $[t]_{\equiv_T} = \{$t' | (t,t') $\in \equiv_T\}$
- $p^\bullet = \{t \in T \mid Pre(p,t) \neq \emptyset\}$
- ${}^\bullet p = \{t \in T \mid Post(t,p) \neq \emptyset\}$
- $t^\bullet = \{p \in P \mid Post(t,p) \neq \emptyset\}$
- ${}^\bullet t = \{p \in P | Pre(p,t) \neq \emptyset\}$
- $(p^\bullet)^\bullet = \bigcup_{t \in p^\bullet} t^\bullet$
- ${}^\bullet({}^\bullet p) = \bigcup_{t \in {}^\bullet p} ({}^\bullet t)$
- $([p]^\bullet_{\equiv_P})^\bullet = \bigcup_{p' \in [p]_{\equiv_P}} (p'^\bullet)^\bullet$

- ${}^\bullet({}^\bullet[p]_{\equiv_P}) = \bigcup_{p' \in [p]_{\equiv_P}} ({}^\bullet({}^\bullet p'))$
- $Var(t) = \{v_i \mid v_i \in \mathcal{G}(t) \vee v_i \in Var(Pre(.,t)) \cup Var(Post(t,.))$

Definition 11 (Buffer place) *A buffer place in a TC-SBP Petri Net model is a place $bp \in B$ where all its related places are equivalent, i.e., iff $\forall$ p' $\in (bp^\bullet)^\bullet$: $(bp^\bullet)^\bullet = [p']_{\equiv_P}$.*

Definition 12 (Well-formed TC-SBP Petri Net model) *A TC-SBP Petri Net model is called well-formed iff:*

1. *each set of equivalent places is preceded by only one buffer place in $\mathcal{B}$, if $[p]_{\equiv_P} \neq \{p\}$ then ${}^\bullet({}^\bullet[p]_{\equiv_P}) = \{bp\} \wedge bp$ is a buffer place in $\mathcal{B}$,*
2. *each variable in a guard function of a transition has a type in Σ, i.e., $\forall\ t \in T$, $\forall\ v \in Var(\mathcal{G}(t))$, $Type(v) \in \Sigma$,*
3. *each arc expression between a place p and a transition tr has the type of the place p and each variable in the arc expression has a type in Σ.*
   - *$\forall\ p \in P$, $\forall\ t \in T$, iff Pre(p, t) $\neq \emptyset$: Type(Pre(p, t)) $= \sigma(p)_{TMS} \wedge \forall\ v \in Var(Pre(p,t))$, $Type(v) \in \Sigma$,*
   - *$\forall\ p \in P$, $\forall\ t \in T$, iff Post(t, p) $\neq \emptyset$: Type(Post(t, p)) $= \sigma(p)_{TMS} \wedge \forall\ v \in Var(Post(p,t))$, $Type(v) \in \Sigma$,*
4. *each variable in an output arc expression of a transition tr appears in at least one of its input arcs expressions, i.e., $\forall\ t \in T \forall\ p \in t^\bullet$, Var(Post(t,p)) $\subseteq$ Var(Pre(.,t)) with Var(Pre(.,t)) $= \cup_{p' \in {}^\bullet t}$ Var(Pre(p',t)).*

4.3.2 Behavior Modeling

In order to define our TC-SBP marking, we use the so-called Timed Multi-set presented in [Jensen 2009] in which elements are specified with their number of appearances and their timestamps. An element in a timed multi-set represents the data value of a request and its timestamps represents the request age (precessing/waiting time). In the following we present the definition of TC-SBP marking.

Definition 13 (TC-SBP marking) *Let N be TC-SBP petri net model. A marking M on N is a function that represents each place as a timed multi-set of tokens matching the type of the place, i.e., $M(p) \in \sigma(p)_{TMS}$. The marking is also extended to equivalent classes, i.e., $M([p]) = {}^{+++}\sum_{p' \in [p]} M(p')$. The set of all markings over N is denoted by M(N).*

Example 2 *Let D be a set of colors representing the data values of tokens, and let tm_A=(1'd_1:0 + 2'd_3:5 + 3'd_2:3) be a timed multi-set over D. The timed multi-set tm_A denotes the marking of the place A in the TC-SBP petri net model. It indicates that A contains one token with data value d_1 and having an age equal to 0, two tokens with data value d_3 and have been in the place for 5 time units, and three tokens with data value d_2 and an age equal to 3.*

Definition 14 (TC-SBP net system) *A Time Colored-Service based Process (TC-SBP) net system is a pair $S = \langle N, M\rangle$ where N is a TC-SBP Petri Net model and M is one of its marking. The marking of N represents all the distributions of requests over the set of services composing the SBP and the set of load balancers while a TC-SBP net system models a particular distribution of requests.*

As stated above, a token models a service request carrying a data value and having an age. The data value is represented by a triplet (c, s, t) where c represents a request category, s represents a request size, and t represents its state. A request might have one of the following states: waiting (0), under-processing (1) or finished (2). A request in a buffer place is always considered in a waiting state. When a request gets in a service place, its state will be either waiting or under-processing depending on the fullness of the service engine, *i.e.*, if it has some capacity left to be used by the current request. Each request stays in under-processing state for a certain amount of time determined by the complexity function of the service and its processing speed given the data size of the request. After spending the amount of time needed to process the request, its state changes to finished indicating its availability to be transferred to the next services in the process. A function called $Finished$ is used to provide if a given token has been processed in the service engine. Therefore, a token can be considered either as: *Available* or *Unavailable*, depending on its state and its age. The token (d, a) is available in a place p under a marking M if it is in finished state and its age a is within the time interval $[t_1, t_2]$ of the place p, *i.e.*, $(d, a) \in M(p) \wedge a \in [t_1, t_2]_p$. It is considered an unavailable token either when its age is greater than the upper bound of the place time interval or when its processing has not been started/finished yet.

Definition 15 (Available/Unavailable marking) *Let M be a marking and p be a place. An available marking (resp. unavailable marking) is a function Avail: $\sigma(p)_{TMS} \rightarrow \sigma(p)_{TMS}$ (resp. Unavail) that returns a sub-timed multiset of tokens from the timed multi-set M(p) such that the age of tokens is within the time boundary of the place p and their state is finished. , i.e., $Avail(M(p)) \subseteqq M(p) \wedge \forall (d, a) \in Avail(M(p)) : a \in \mathcal{I}(p) \wedge Finished((d, a))$ (resp. $Unavail(M(p)) \subseteqq M(p) \wedge \forall (d, a) \in Unavail(M(p)) : a \notin \mathcal{I}(p) \vee \neg Finished((d, a))$).*

The available marking, returned by the function $Avail$, is used for enabling transitions in the set T in the TC-SBP net system S=$\langle N, M\rangle$ that have to be fired in the marking M. The firing of transitions changes the net system S to S' by taking available

tokens from the input places of transitions and adding them to their output places which changes the marking M to M'. Thus, by firing a transition t, tokens from the available marking of the input places of t that verify the guard expression $\mathcal{G}(t)$ and match the type of the output places of t will be transferred to the corresponding output places.

Definition 16 (Binding transition) *Let t be a transition. A binding b of t is a function that associates each variable v in Var(t) to a value b(v) such that $b(v) \in Type(v)$. We write a a binding as: $b = \langle v_1 = val_1, v_2 = val2, \ldots, v_n = val_n \rangle$, where v_i is a variable in Var(t) and val_i is the bounded value to v_i. The set of all bindings for a transition t is denoted B(t).*

The result of evaluating the guard expression $\mathcal{G}(t)$ of a transition t in a binding b can be denoted as $\mathcal{G}(t)\langle b \rangle$. Similarly, we denote by $Pre(p,t)\langle b \rangle$ (resp. $Post(t,p)\langle b \rangle$) the result of the evaluation of the expression Pre(p,t) (resp. Post(t,p)) of the arc connecting the place p to the transition tr in the binding b. $Pre(p,t)\langle b \rangle$ (resp. $Post(t,p)\langle b \rangle$) is a timed multiset over $\sigma(p)$.

Definition 17 (Fireable transition) *Given a TC-SBP net system $S = \langle N, M \rangle$ and a transition t, we say that a transition t is fireable in the marking M, noted by $M[t\rangle^b$ iff we can find a binding b such that the following properties hold true:*

1. $\mathcal{G}(t)\langle b \rangle = true$,
2. $\forall p \in^\bullet t, Pre(p,t)\langle b \rangle \lll= Avail(M(p))$.

A class of transitions is fireable in M, $M[[t]\rangle^b$, iff $\exists\ t' \in [t] : M[t'\rangle^b$

Definition 18 (Firing transition) *Let M be a marking and t be a transition, the firing of t using a binding b changes the marking M to M' s.t. $\forall\ p \in P$: M'(p) = $Unavail(M(p)) +\!+\!+ [(Avail(M(p)) -\!-\!- Pre(p,t)\langle b \rangle) +\!+\!+ Post(t,p)\langle b \rangle]$. We denote the firing by $M[t\rangle^b M'$. The transition notation is extend to classes using $M[[t]\rangle^b M'$ where $M' \in \{M" | \exists t' \in [t] : M[t'\rangle^b M"\}$.*

$$M'(p) = \begin{cases} Unavail(M(p)) +\!+\!+ [Avail(M(p)) -\!-\!- Pre(p,t)\langle b \rangle] & iff\ Pre(p,t) \neq \emptyset \\ Unavail(M(p)) +\!+\!+ [Avail(M(p)) +\!+\!+ Post(t,p)\langle b \rangle] & iff\ Post(t,p) \neq \emptyset \end{cases}$$

Example 3 *Let's take the example of SBP presented in Section 1.3 to illustrate our TC-SBP net system. Figure 4.2 represents the corresponding TC-SBP Net system of SBP for MER. Each place in the petri net model is annotated with its type, its time complexity function C, its capacity, its processing speed, its time interval I, the marking M for no empty places and whether the place is elastic or not. The accepted token format for the places is of type CAT×PAIR×INT×INT where CAT is a set of categories which contains three categories c_1, c_2 and c_3 respectively for the small, medium and large sequence files (request), PAIR represents a couple of integer values (N, L) for the number of sequences in the file and the length of sequences, INTxINT represents the state and the age of the*

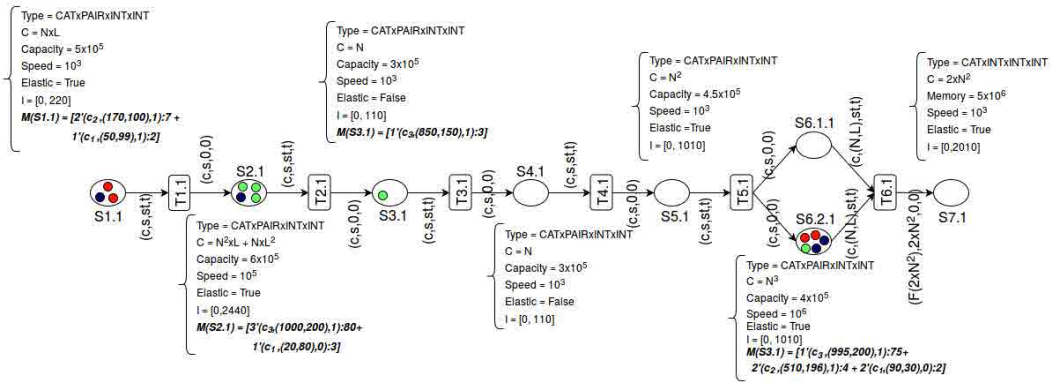


Figure 4.2: TC-SBP petri net system of MER process

token. For example, the service place $S2.1$ *corresponding to the service* $S2$ *in Figure 1.2, is of time complexity* $O(N^2 \times L + N \times L^2)$*. We assume that its corresponding service engine has a capacity* 6×10^5 *(quantity of data that the service engine can process) and can process* 10^5 *instructions per time unit and it considers any request that has been in it for more than 2440 time units as an outdated request. We assume that at time* t_1 *the service engine associated to the place* $S2.1$ *holds four requests: one waiting request* (c_1, *(20,80), 0) of category* c_1 *with data size (20,80) and an age equal to 3, three under-processing requests with data value* (c_2, *(1000,200), 1) and an age 20.*

We assume in our example that, after a period of time, certain requests will overcome the maximum response time leading to loss of QoS in some services. For the clarity of the example, we will focus specially on service $S2.1$ *to illustrate the application of Duplication/Consolidation operations that we present in the following section.*

4.4 Elasticity Operations

An execution environment of a SBP evolves over time (changes its structure) according to requests load by adding/removing service engine copies in order to meet its QoS requirements (*e.g.*, maximum response time). Elasticity operations enable acquiring and releasing resources. Elasticity strategies govern the application of these operations.

In our work, elasticity operations operate on service and not on the entire process level. That means these operations can be applied on component services separately as dictated by the used strategy. The initial execution environment contains one service engine for each service involved in the SBP. In the following, we present the Duplication and Consolidation operators enabling to implement the infrastructure elasticity.

4.4.1 Duplication Operator

The duplication operator D(S,p, [δ, θ, η]) allows to create a new copy of a service engine in a TC-SBP net system. The created copy might have a new configuration, *i.e.*

a new request category δ, a new capacity θ and/or a new processing speed η. Thus the duplication operator takes as input (i) a TC-SBP net system S = $\langle N, M\rangle$ that will be changed to a new TC-SBP net system S' = $\langle N', M'\rangle$ by applying the operator, (ii) a place p representing the service engine to be duplicated, and (iii) optionally a new service engine configuration [δ, θ, η].

The duplication of p is performed either by connecting (i) two places $p^c_{\delta\theta\eta}$ and bp to the current net where the first place is the new copy of p with a color set δ, a capacity θ and a processing speed η, and the latter is a buffer place in B to be shared between p and $p^c_{\delta\theta\eta}$ if this is the first duplication of the initial service engine, or (ii) by adding only the new copy $p^c_{\delta\theta\eta}$ otherwise.

Definition 19 (Duplication Operator) *Let $S = \langle N, M\rangle$ be a TC-SBP net system, p be a place in S and $[\delta, \theta, \eta]$ be a new configuration. The duplication operator D(S,p, [$delta, \theta, \eta$]) changes the status of the TC-SBP net system S to a new TC-SBP net system $S' = \langle N', M'\rangle$ where:*

- $\Sigma' = \Sigma$
- $P' = S' \cup B'$ *with* $S' = S \cup \{p^c_{\delta\theta\eta}\}$ *and* $B' = B \cup \{bp\}$ *iff* $\mid [p]_{\equiv_P} \mid = 1$
- $T' = T \cup T''$ *with*

$$T'' = \begin{cases} (i) \quad \{t, t' \mid t \text{ is a new transition } \wedge t' = \eta(t)\} & \\ \qquad \cup \{t^c \mid t \in p^\bullet \wedge t^c = \eta(t)\} & if \mid [p]_{\equiv_P} \mid = 1 \\ (ii) \quad \{t^c \mid t \in ({}^\bullet p \cup p^\bullet) \wedge t^c = \eta(t)\} & otherwise \end{cases}$$

  where $\eta(t)$ generates a new copy of t which is not in T.

- σ': $P' \rightarrow \Sigma'$ *with* $\sigma'(p') = \sigma(p')$ *for all* $p' \in P$ *and* $\sigma'(p^c_{\delta\theta\eta}) = \delta$.
- *Capacity'*: $P' \rightarrow \mathbb{N}$ *with Capacity'(p') = Capacity(p') for all* $p' \in P$ *and Capacity'*$(p^c_{\delta\theta\eta})$ $= \theta$.
- *Speed'*: $S' \rightarrow \mathbb{N}$ *with Speed'(p') = Speed(p') for all* $p' \in S$ *and Speed'*$(p^c_{\delta\theta\eta}) = \eta$.
- *Elastic'*: $S' \rightarrow BOOLEAN$ *with Elastic'(p') = Elastic(p') for all* $p' \in S$ *and Elastic'*$(p^c_{\delta\theta\eta})$ *= Elastic(p).*
- $V' = V$.
- $\mathcal{C}'$: $S' \rightarrow EXPR$ *with* $\mathcal{C}'(p') = \mathcal{C}(p')$ *for all p' in S and* $\mathcal{C}'(p^c_{\delta\theta}) = \mathcal{C}(p)$
- $\mathcal{G}'$: $T' \rightarrow EXPR$ *with* $\mathcal{G}'(t')$=$\mathcal{G}(t')$ *for all* $t' \in T$ *and* $\mathcal{G}'(t^c) = \mathcal{G}(t) \mid t \in T \wedge t^c \in T$" $\wedge\, t^c = n^{-1}(t)$
- *Pre'*: $P' \times T' \rightarrow EXPR$
- *Post'*: $T' \times P' \rightarrow EXPR$

- $\equiv_{P'} \subseteq P' \times P'$ *with* $\equiv_{P'} = \equiv_P \cup \{(p, p^c_{\delta\theta})\}$. *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.*

- $\mathcal{I}'$*: P' →* I *with* $\mathcal{I}'$*(p') =* $\mathcal{I}$*(p') for all p' ∈ P and* $\mathcal{I}'(p^c_{\delta\theta}) = \mathcal{I}(p)$.

- *M': a marking that maps each place p'∈ P' into a timed multi-set of tokens in* $\sigma(p')_{TMS} \rightarrow \Sigma_{TMS}$ *with*

$$M'(p') = \begin{cases} M(p') & if\ p' \neq p^c_{\delta\theta}\ \wedge\ \neg(p' = bp \wedge \mid [p]_{\equiv_P} \mid = 1) \\ \emptyset_{\sigma(p')} & otherwise \end{cases}$$

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

$$\text{Pre'(p',t')} = \begin{cases} Pre(p', t') & if\ p' \in P \wedge t' \in T \\ Pre(p', t) & if\ t \in T \\ & \wedge\ t' \in (T' \setminus T) \\ & \wedge\ t' \in [t]_{\equiv_{T'}} \\ & \wedge\ p' \in (P \setminus \{p\}) \\ Pre(p, t) & if\ t \in T \\ & \wedge\ t' \in (T' \setminus T) \\ & \wedge\ t' \in [t]_{\equiv_{T'}} \\ & \wedge\ p' = p^c_{\delta\theta} \\ x & if\ p' = bp \\ & \wedge\ \mid [p]_{\equiv_P} \mid\ = 1 \\ & \wedge\ t' \in (T' \setminus T) \\ & \wedge [t']_{\equiv'_T} \cap T = \emptyset \\ \emptyset & otherwise \end{cases}$$

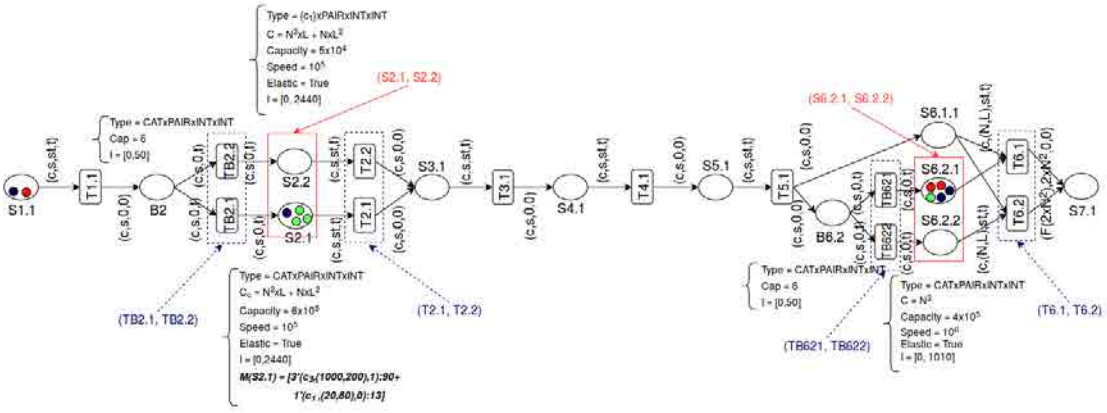


Figure 4.3: Application of Duplication operation on service engines S2.1 and S6.2.1 of MER process

$$
\text{Post'(t',p')} = \begin{cases}
Post(t', p') & if\ p' \in P \setminus \{p\} \\
 & \wedge\ t' \in T \\
Post(t, p') & if\ t \in T \\
 & \wedge\ t' \in (T' \setminus T) \\
 & \wedge\ t' \in [t]_{\equiv_{T'}} \\
 & \wedge\ p' \in (P \setminus \{p\}) \\
 & \wedge\ p' \notin [p]_{\equiv'_P} \\
Post(t', p) & if\ t' \in T \\
 & \wedge\ |\, [p]_{\equiv_P} \,| = 1 \\
 & \wedge\ p' = bp \\
Post(t, p) & if\ t \in T \\
 & \wedge\ t' \in (T' \setminus T) \\
 & \wedge\ t' \in [t]_{\equiv_{T'}} \\
 & \wedge\ |\, [p]_{\equiv_P} \,| \neq 1 \\
 & \wedge\ p' = p^c_{\delta\theta} \\
x & if\ (p' = p \vee p' = p^c_{\delta\theta}) \\
 & \wedge\ |\, [p]_{\equiv_P} \,| = 1 \\
 & \wedge\ t' \in (T' \setminus T) \\
 & \wedge\ [t']_{\equiv'_T} \cap T = \emptyset \\
\emptyset & otherwise
\end{cases}
$$

In the following, we present an illustrative example for applying this operator.

Example 4 *Let's take the TC-SBP net system state presented in Figure 4.2. We assume that after some period of time the request of category* c_1 *in service engine* $S2.1$ *has been waiting for too long which led it to overcome the maximum response time defined as QoS requirement. Using an elasticity strategy for hybrid scaling, the service* $S2.1$ *is*

duplicated to create a new copy of the service engine represented by the place $S2.2$ for requests in the category c_1 with new capacity specified for small requests. Figure 4.3 is the resulting system for applying a duplication operation on the service $S2.1$ in Figure 4.2. We remark the adding of two places to the initial TC-SBP net system where the first one is a copy of the service $S2.1$ contained an empty set of tokens and the other is a buffer place $B2$ representing the load balancer of $S2.1$ with a queue length equal to 6. At the same time, the duplication operation is also applied on the service engine $S6.2.1$ that we assume it has violated the specified QoS. So, the application of the operation creates a new copy of the service engine $S6.2.1$ and a buffer place $B6.2$ connecting them. The pairs indicated in Figure 4.3, e.g., (S2.1, S2.2) and (T6.1, T6.2), represent places or transitions connected by an equivalence relation (i.e., $(S2.1, S2.2) \in \equiv_P$ and $(T6.1, T6.2) \in \equiv_T$)

4.4.2 Consolidation Operator

The consolidation operator C(S,p,p^c, bp) removes a service copy in a TC-SBP net system. The consolidation of a copy p^c of a given service engine place p is performed either (i) by removing only the place p^c from S if there will remain p and another copy, or (ii) by removing the place p^c and releasing its related buffer place bp otherwise. An illustrative example is presented in the following.

Definition 20 (Consolidation Operator) *Let $S = \langle N, M\rangle$ be a TC-SBP net system and let p, p^c, bp, be places in N with $(p,p^c) \in \equiv_P \wedge\ p \neq p^c$ and bp is a buffer place in B shared between p and p^c. The consolidation operator C(S,p,p^c, bp) changes the status of the TC-SBP net system S to a new TC-SBP net system S'=$\langle N', M'\rangle$ where:*

- $\Sigma' = \Sigma$
- *P' = S' ∪ B' with S' = S \ {p^c} and B' = B \ {bp} iff* $|\ [p]_{\equiv_P}\ | = 2$
- *T' = T \ T'' with*

  $$T'' = \begin{cases} (p^c)^\bullet \cup [^\bullet p^c] & if\ \ |\ [p]_{\equiv_P}\ | = 2 \\ (p^c)^\bullet \cup^\bullet p^c & otherwise \end{cases}$$

- *σ': P' → Σ' with σ'(p') = σ(p') for all p' ∈ P'*
- *Capacity': P' → ℕ with Capacity'(p') = Capacity(p') for all p' ∈ P'*
- *Speed': S' → ℕ with Speed'(p') = Speed(p') for all p' ∈ S'*
- *Elastic': S' → BOOLEAN with Elastic'(p') = Elastic(p') for all p' ∈ S and Elastic'($p^c_{\delta\theta\eta}$) = Elastic(p).*
- *V' = V*
- *C': S' → EXPR with C'(p') = C(p') for all p' ∈ S'*

- $\mathcal{G}'$: $T' \rightarrow EXPR$ *with* $\mathcal{G}'(t') = \mathcal{G}(t')$ *for all* $t' \in T'$
- *Pre'*: $P' \times T' \rightarrow EXPR$
- *Post'*: $T' \times P' \rightarrow EXPR$
- $\equiv_{P'} \subseteq P' \times P'$ *with* $\equiv_{P'} = \equiv_P \setminus \{(p, p^c)\}$.
- $\equiv_{T'} \subseteq T' \times T'$ *with* $\equiv_{T'} = \equiv_T \setminus \{(t, t^c) | t^c \in T"\}$.
- $\mathcal{I}'$: $P' \rightarrow I$ *with* $\mathcal{I}'(p') = \mathcal{I}(p')$ *for all* $p' \in P'$
- *M': a marking that maps each place* $p' \in P'$ *into a timed multi-set of tokens in* $\sigma(p')_{TMS} \rightarrow \Sigma_{TMS}$ *with:*

$$M'(p') = \begin{cases} M(p) + + + M(p^c) + + + M(bp) & if\ p' = p \\ & \wedge\ |\ [p]_{\equiv_P}\ | = 2 \\ M(p) + + + M(p^c) & if\ p' = p \\ M(p') & otherwise \end{cases}$$

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

$$\text{Post'(t',p')} = \begin{cases} Post(t', p') & if\ p' \in P' \setminus \{p\} \\ & \wedge\ t' \in T' \\ Post(t', bp) & if\ t' \in T' \wedge p' = p \end{cases}$$

$$\text{Pre'(p',t')} = \text{Pre(p',t')} \mid p' \in P' \wedge t' \in T'$$

Example 5 *Figure 4.4(b) illustrates the consolidation of the service* $S2.1$ *by releasing its copy* $S2.2$ *along with their associated load balancer (i.e.,* $B2$*) to change the state of the net system from the state in Figure 4.4(a) to the state presented in Figure 4.4(b). The red section in the Figure 4.4(a) represents the removed components when the consolidation operation is applied and the blue arc is the added component that reconnects the transition* $t1.1$ *to the service place* $S2.1$*. Before removing the buffer place* $B2$*, the requests that it contains are transferred from* $B2$ *to the service place* $S2.1$ *in which they will be processed.*

4.5 Conclusion

In this chapter, we presented a formal model, based on timed-colored Petri net, for describing elastic execution environments of SBPs. Contrary to similar work, our model enables (*i*) the description of service engines characteristics that allows the simulation of processing data-based requests, (*ii*) the description of requests characteristics which allows them to be distinguished from each other, and (*ii*) the adding of a load balance to

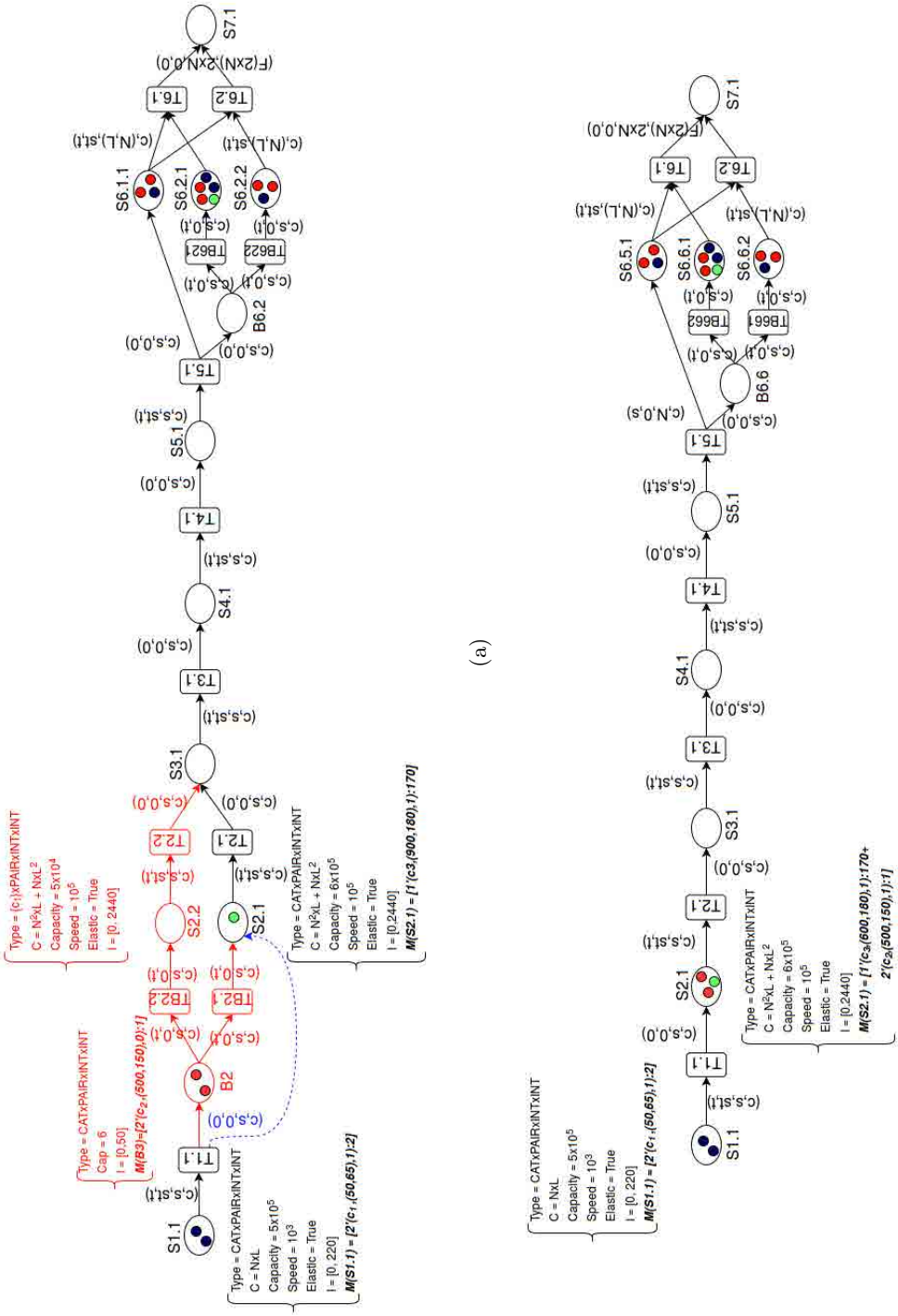


Figure 4.4: Application of Consolidation operation on service engine S2.1 of MER process

balance the load between several service copies. Our model allowed thereafter defining new elasticity operations such as hybrid scaling and considering additional metrics when defining elasticity strategies.

We tackle in the next chapter the problem of defining elasticity strategies for SBPs by proposing two linked domain-specific language (DSL) that allow describing different strategies for different elasticity models.

CHAPTER 5

Description of Elasticity Strategies for Elastic SBPs

Contents

5.1 Introduction

Elasticity strategies are policies that are used to manage elasticity by deciding when, where and how to use elasticity capabilities/actions (*e.g.*, adding or removing resources) that are defined in the elasticity model of the managed system. Usually, an elastic system is managed by a controller that implements a specific elasticity model and uses an elasticity strategy to control the system adjustment decisions. When defining a

strategy, only the elasticity capabilities that could be performed by the controller of the elastic system should be allowed.

In this chapter, we propose two domain-specific languages for elastic SBPs that allow together to describe different elasticity strategies for different elasticity models. Our first language, named StratModel, is a descriptive language for describing elasticity models. It permits users to define different elasticity models, with different elasticity capabilities/actions and customized monitoring metrics, and to generate their associated elasticity controllers in order to use them to evaluate elasticity strategies on a SBP model. Our second language, named Strat, is a rule-based language for describing elasticity strategies. It is proposed relied on StratModel to define the elasticity model on which its grammar will be adapted. This chapter is organized as follows. In Section 5.2, we introduce the concept of domain-specific language. Then, we present our languages design in Section 5.3. Thereafter, we describe the details of StratModel language in Section 5.4, followed by the details of Strat language in Section 5.5.

5.2 Domain specific language

A Domain Specific language (DSL) is defined as a small and usually declarative computer language that provides, through appropriate abstractions and notations, expressive power for a particular problem domain contrary to a General Purpose Language (GPL) which is broadly applicable for any kind of software problem, like Java, C++ [Deursen 2000]. DSLs have been around for almost as long as computing has existed. Some examples of DSLs are APT that was developed in 1957-1958 for generating instructions for numerically controlled machine tools [Ross 1978]; BNF which is a syntax specification formalism developed in 1959 [Backus 1959]; SQL, a DSL for handling persistent data developed in the early 1970s; VHDL that has been first introduced in 1983 as a VHSIC Hardware Description Language, and many other widely used DSLs.

The key advantages of DSLs compared to other GPLs reside in the fact that they provide significant gains in expressiveness and ease of use for a particular problem domain which in turn lead to gains in productivity and reduced maintenance costs [Mernik 2005]. Usually, domain experts are not familiar with GPLs to solve their domain problems. So, it is more suitable for them to address their domain problems through a DSL that represents the abstractions of their own domain knowledge. Even for professional trained developers, using DSLs within a software development project brings a lot of benefits like increasing flexibility, reliability, productivity, and usability as proven through evaluating many case studies [Czarnecki 2000, Mernik 2005, Wile 2004]. In this chapter, we present two DSLs that are proposed to facilitate the description of elasticity strategies for different elasticity models.

5.3 Languages design

During designing our domain-specific language for describing elasticity strategies for SBPs, we went from providing a language that is specified for a particular elasticity model to a language that can be adapted to any elasticity model. At first, we proposed a rule-based DSL named STRAT that allows to define elasticity rules only for three main actions that provide horizontal elasticity, namely, (*i*) `duplicate` that creates a new copy of an overloaded service in order to meet its workload increase, (*ii*) `consolidate` that releases an unnecessary copy of a service in order to meet its workload decrease, and (*iii*) `routing` that controls the way a load of a service is routed over the set of its copies. This was our first attempt in designing our language. However, in this way, the use of our language will be restricted to defining strategies that will be used only by an elasticity controller implementing such elasticity model. So, in order to make it general enough to describe strategies for different elasticity models and allows the adding of new possible actions, two solutions come in mind. The first solution is to provide STRAT language grammar with a constant set of actions used in the commercial cloud-solutions and the research papers. Such solution makes the users (*i.e.*, SBP holders) constrained to a set of pre-defined actions and parameters and does not enable the adaptation to new elasticity capabilities that can be provided by the community.

The second solution, that we adopt in our work, consists in designing another language named STRATModel for describing elasticity models for SBPs and relying STRAT to it. In this way, users are allowed to separately provide the description of an elasticity model on which STRAT language will be based. So, the latter will provide users with only the elasticity capabilities defined in the elasticity model by the elasticity manager. This solution is based on the notion of *"cross-reference"* to create links between the languages. The elasticity capabilities are defined using STRATModel as objects and then their references are used in STRAT language as part of its grammar. Along the elasticity capabilities, STRATModel allows users to define a set of metrics that can be used as QoS metrics or functions called inside STRAT script using their references, and to specify the properties that the users can re-configure when applying an action according to a STRAT strategy. The links created between the two languages only constraints the users with the order of defining their system elements. So, they have to first define an elasticity model using STRATModel on which STRAT will be based and then they can define their elasticity strategies.

In the following sections, we describe the details of STRATModel language and STRAT language with their use in defining respectively the elasticity model and the elasticity strategy of the elastic system given in Section 1.3

5.4 STRATModel: Elasticity Model Description Language

In the following, we present an overview of our STRATModel language, followed by the details of its grammar for describing elasticity models. Thereafter, we detail the

template used to generate elasticity controllers for evaluating elasticity strategies for SBPs.

5.4.1 STARTModel Overview

StratModel language is proposed as a part of StratFram framework for evaluating elasticity strategies for SBPs that we present in the next chapter (*cf.* Chapter 6). It is designed to allow the description of elasticity models and the generation of their corresponding elasticity controllers using a pre-defined elasticity controller template. An elasticity model defines the ground terms and functionalities that describe SBPs elasticity such as the elasticity actions to be undertaken, metrics to monitor to trigger the elasticity actions and properties to access and reconfigure. Hence, it is the basis for specifying elasticity strategies and constructing an elasticity controller that manages and evaluates the elasticity of SBPs. An elasticity controller is used to monitor a SBP and analyse its performance by inspecting an elasticity strategy to decide whether some actions that might reconfigure some properties of the managed component are needed to be applied. So, the monitoring metrics, actions, and properties are the main compositions of an elasticity model in StratModel. A StratModel metric can be defined as either a basic metric, *i.e.*, obtained directly from the monitored component property, or a composed metric. A StratModel action is defined to be applied on a specific type of component of a SBP model, *e.g.*, Service, and make changes on it. StratModel is designed relying on the specification of SBP model which defines the components of modelling elastic SBPs (*cf.*, Chapter 4).

Describing elasticity model in StratModel depends on what kind of information encoded in the SBP models that business process holders want to provide and manage by the generated elasticity controller, and which type of elasticity strategies they want to specify. Such information is required to choose either to include or exclude some functionalities to/from the generated elasticity controller. In the following, we discuss in more details the StratModel grammar used to define elasticity models, followed by how the elasticity controllers are generated from the defined elasticity models.

5.4.2 STRATModel Grammar

The top-level of StratModel specification grammar is given in Grammar 5.1 using the Backus Normal Form (BNF). StratModel documents are composed of two main parts. The first part, *i.e.*, the elasticity model description part, contains the definition of the essential elements to describe an elasticity model. The second part is for defining business process transformation states which specify the transformations that occur on the business process model when applying some defined actions. In the following, we will provide a detailed description of each of these parts.

⟨*ElasticityModel*⟩ ::= ⟨*ModelDescription*⟩ ⟨*ProcessStates*⟩

⟨*ProcessStates*⟩ ::= ⟨*ProcessState*⟩ ⟨*ProcessStates*⟩ | ⟨*empty*⟩

Grammar 5.1: General StratModel Grammar

5.4.2.1 Elasticity Model Description

As shown in Grammar 5.2, an elasticity model in StratModel is mainly composed of two sets of statements, descriptive and functional, encapsulated in a block defined by `ElasticityModel` and identified by a name.

⟨*ModelDescription*⟩ ::= '`ElasticityModel`' ⟨*id*⟩ '{' ⟨*ModelStatements*⟩ '}'

⟨*ModelStatements*⟩ ::= ⟨*GeneralDescription*⟩ ⟨*ItemsDefinitions*⟩

⟨*ItemsDefinitions*⟩ ::= ⟨*ItemDefinition*⟩ ⟨*ItemsDefinitions*⟩ | ⟨*empty*⟩

⟨*ItemDefintion*⟩ ::= ⟨*Action*⟩ | ⟨*Metric*⟩ | ⟨*Property*⟩

Grammar 5.2: Grammar for describing elasticity model in StratModel

The user is allowed to first describe the general aspect of an elasticity model. As shown in Grammar 5.3, it consists on providing a reference ID with optionally a reference to the managed component. The managed component refers to a model for elastic execution environment of a SBP described using a domain specific language called SBP, *cf.* Section 6.2.2. In addition, the user can specify whether to use a default routing mechanism or a customized one specified as a StratModel action. Depending on whether a temporal information is a part of SBP models the business process holders want to manage, it is essential to indicate if a timer is needed to be included in the elasticity controller implementing in order to update the value of the time related attributes in the model. Also, some elasticity strategies may need to use previously gathered data of monitored metrics to make elasticity decisions. Such data are stored in a knowledge base. So, the user may also indicate the use of a knowledge base and the frequency of monitoring. Thereafter, the functional statements are provided to define the essential elements for describing the functionalities of the elasticity controller implementing the elasticity model. They are divided into actions to be undertaken, metrics to monitor and properties to access and to reconfigure.

Example 6 *Let's take the elasticity model for hybrid scaling that performs two main*

⟨*GeneralDescription*⟩ ::= ⟨*Reference*⟩ ⟨*ManagedComponent*⟩ ⟨*Routing*⟩ ⟨*Timer*⟩ ⟨*KnowledgeBase*⟩ ⟨*Frequency*⟩

⟨*Reference*⟩ ::= '***referenceID***' ':' ⟨*id*⟩

⟨*ManagedComponent*⟩ ::= '***managedComponent***' ':' [SBP::SBPModel] | ⟨*empty*⟩

⟨*Routing*⟩ ::= '***routing***' ':' ⟨*RT*⟩ | ⟨*empty*⟩

⟨*Timer*⟩ ::= '***timer***' ':' ⟨*boolean*⟩ | ⟨*empty*⟩

⟨*KnowledgeBase*⟩ ::= '***knowledgebase***' ':' ⟨*Boolean*⟩ | ⟨*empty*⟩

⟨*Frequency*⟩ ::= '***frequency***' ':' ⟨*int*⟩ | ⟨*empty*⟩

⟨*RT*⟩ ::= 'DEFAULT' | 'GENERATED'

Grammar 5.3: Grammar for general description of an elasticity model in StratModel

elasticity actions namely 'Duplication' and 'Consolidation as described in Section 1.3. We use our SBP model for molecular evolution reconstruction as the managed component. The elasticity strategies that will be defined and used are reactive and they do not need a knowledge base for making elasticity decisions. Listing 5.1 provides a general description of the elasticity model named 'ElasticityModel1'.

```
ElasticityModel ElasticityModel1 {
  referenceID : 'ElasticityModel1'
  managedComponent : MERProcess
  routing : DEFAULT
  timer : true
  knowledgebase : false
  frequency : 3
  ...
}
```

Listing 5.1: Example of describing an Elasticity Model with StratModel

- ***Action***: A StratModel ***action*** is defined by a set of statements for describing its functionality and details that are used to generate its implementation mechanism. As given in Grammar 5.4, it is defined by a name, a reference ID and a component. The latter is used to specify the type of elements on which the action can be applied, *i.e.*, ***Process***, ***Service*** or ***Router***. For instance, a Routing action can be defined for ***Router*** component to allow to control the execution flow of

requests between SBP's services. The keywords *delay* and *multiple* are used to specify respectively the time delay of applying the action and whether its multiple application is allowed.

⟨*Action*⟩ ::= '*action*' ':' ⟨*ActionStatements*⟩ ';'

⟨*ActionStatements*⟩ ::= ⟨*Name*⟩ ⟨*Reference*⟩ ⟨*Component*⟩ ⟨*Delay*⟩ ⟨*Multiple*⟩ ⟨*Transformation*⟩

⟨*Name*⟩ ::= '*name*' ':' ⟨*string*⟩

⟨*Reference*⟩ ::= '*referenceID*' ':' ⟨*id*⟩

⟨*Delay*⟩ ::= '*delay*' ':' ⟨*int*⟩ | ⟨*empty*⟩

⟨*Multiple*⟩ ::= '*multiple*' ':' ⟨*Boolean*⟩ | ⟨*empty*⟩

⟨*Transformation*⟩ ::= '*cases*' ':' ⟨*Examples*⟩

⟨*Examples*⟩ ::= ⟨*Example*⟩ ⟨*Examples*⟩ | ⟨*empty*⟩

⟨*Example*⟩ ::= '*apply*' '*on*' ⟨*Elements*⟩ '*transform*' ⟨*id*⟩ '*to*' ⟨*id*⟩

⟨*Elements*⟩ ::= ⟨*string*⟩ ',' ⟨*Elements*⟩ | ⟨*string*⟩

Grammar 5.4: Grammar for describing elasticity actions in StratModel

Applying the defined action on the managed SBP model changes its structure. It transforms it from one state to another. This transformation can be described in StratModel as transformation cases. A transformation case is specified through giving an example of an initial state of the SBP model and the resulting state after applying the action (*cf.* Section 5.4.2.2). The idea of using examples to define the transformations on SBP model follows the *by-example paradigm* [Cypher 1993] that allows the software to drive information from a set of examples that specify how things are done or what the user expects. The most prominent approaches for *by-example paradigm* are *Programming by-example* [Lieberman 2001] which permits to create a program from user's actions recorded as replayable macros, and *Query by-example* [Zloof 1975] which has been developed for querying database systems by allowing end-users to provide examples of query results. These approaches allow to use examples in some way to overcome the complexity of selected problems in the field of computer science. In our work, we argue that giving examples of transformations can be more friendly for business process holders than providing complex formal transformations instructions. So, in StratModel, the user is allowed to give a set of examples that describe different cases of applying the action on specific elements.

Example 7 *In the following, we provide the description of the duplication action in the elasticity model that we have presented in Section 1.3. This action is defined to be applied on service engines components. Its execution will be delayed by 4 time units. There are two transformation cases. The first case is when the duplication action will be applied for the first time on a service engine to create a new copy and a service engine load balancer. The second case is when there already exists more than one service copy sharing a load balancer in the process model. The description of the action is given in Listing 5.2.*

```
ElasticityModel ElasticityModel1 {
  ...
  action :
    name : 'Duplicate'
        referenceID : 'D'
        component : Service
        delay : 4
        cases:
          apply on 'S2' transform state1 to state2
          apply on 'S2' transform state3 to state4
  ;
  ...
}
```

Listing 5.2: Example of describing an action with StratModel

- ***Metric***: A StratModel `metric` is identified by a name and has a low-level reference ID. It is associated to an entity that can be a process, a service, a load balancer or a requests. It can also be obtained for a specific group of requests by allowing grouping and may have a defined unit of measure. A StratModel metric can be either a basic metric obtained from a low-level property or a composite metric whose values are defined by a mathematical expression involving other basic or composite metrics. For example, the metric `ExecutionTime` is defined as a basic metric that refers to the age of a service request. When specifying composite metrics, `expression` is used to define how the value is computed. A StratModel metric can also be obtained for a specific group of requests by indicating `group` as true. The StratModel metric specification is given by Grammar 5.5 using also the Backus Normal Form (BNF).

Example 8 *Listing 5.3 illustrates the description of a metric named 'waitingTime' which captures the waiting time of a request in a service. It is the subtraction of the values of two metrics: 'processingTime' and 'executionTime' which refers to a request attribute named 'age'.*

⟨*Metric*⟩ ::= '**metric**' ':' ⟨*MetricStatements*⟩ ';'

⟨*MetricStatements*⟩ ::= ⟨*Name*⟩ ⟨*Reference*⟩ ⟨*Entity*⟩ ⟨*OnGroups*⟩ ⟨*Unit*⟩ ⟨*MetricExpression*⟩

⟨*Name*⟩ ::= '**name**' ':' ⟨*string*⟩

⟨*Reference*⟩ ::= '**referenceID**' ':' ⟨*id*⟩

⟨*Entity*⟩ ::= '**level**' ':' ⟨*MetricLevel*⟩

⟨*MetricLevel*⟩ ::= '**Service**' | 'LoadBalancer' | 'Process' | 'Request'

⟨*OnGroups*⟩ ::= '**group**' ':' ⟨*Boolean*⟩ | ⟨*empty*⟩

⟨*Unit*⟩ ::= '**unit**' ':' ⟨*string*⟩ | ⟨*empty*⟩

⟨*MetricExpression*⟩ ::= '**expression**' ':' ⟨*Expression*⟩

⟨*Expression*⟩ ::= ⟨*Exp*⟩ ⟨*MathOp*⟩ ⟨*Expression*⟩ | [Metric] | ⟨*Double*⟩

⟨*Exp*⟩ ::= '(' ⟨*Expression*⟩ ')' | [Metric] | ⟨*Double*⟩

⟨*MathOp*⟩ ::= '+' | '-' | '*' | '/'

Grammar 5.5: Grammar for describing metrics in StratModel

```
ElasticityModel ElasticityModel1 {
  ...
  metric :
    name : 'WaitingTime'
        level : Request
        expression : executionTime − processingTime
  ;
  metric :
        name : 'executionTime'
        referenceID : 'age'
        level: Request
 ;
  ...
}
```

Listing 5.3: Example of describing metrics with StratModel

- ***Property***: In elasticity model, some defined actions may require to access or modify/adjust some low-level properties of the managed SBP and its services. So, the user has to be able to define those properties and whether they are configurable or not. A StratModel ***property*** is primary defined by a name and a reference.

⟨*Property*⟩ ::= '**property**' ':' ⟨*PropertyStatements*⟩ ';'

⟨*PropertyStatements*⟩ ::= ⟨*Name*⟩ ⟨*Reference*⟩ ⟨*Config*⟩

⟨*Name*⟩ ::= '***name***' ':' ⟨*string*⟩

⟨*Reference*⟩ ::= '***referenceID***' ':' ⟨*id*⟩

⟨*Config*⟩ ::= '**configurable**' ':' ⟨*Boolean*⟩

Grammar 5.6: Grammar for describing properties in StratModel

Example 9 *As we indicated in our example in Section 1.3, there are two properties that are accessible and reconfigurable by the duplication action. Listing 5.4 shows how they are specified using* StratModel.

```
ElasticityModel ElasticityModel1 {
  ...
  property :
        name : 'cap'
        referenceID : 'capacity'
        config : true
  ;
  property :
        name : 'cat'
        referenceID : 'groups'
        config : true
  ;
  ...
}
```

Listing 5.4: Example of describing properties with StratModel

5.4.2.2 Business process transformation state definition

As previously stated, applying an action on a SBP model transforms it from one state to another. A transformation state represents the managed SBP model at a specific timestamp t. It is described in a block defined by ***ProcessState*** and identified by a name that is used to refer to the state in the action description section (*cf.* Grammar 5.7).

Two ways are allowed to define the state of a SBP model at timestamp t. The first is by specifying the process and its components using specific notation. The block defined by $< SBPModel >$ encapsulates the process general decription, the groups of requests allowed in the process, its services that are split into service engines and load balancer, the routers that connect its services, and the links between services and

routers. A detailed discussion of the grammar used to describe processes in which $< SBPModel >$ is the enter point can be found in Section 6.2.2. The second way to provide the SBP model description at timestamp t, is by specifying the URL of the business process petri net model encoded in the Petri Net Markup Language (PNML).

⟨*ProcessState*⟩ ::= '**`ProcessState`**' ⟨*id*⟩ '{' ⟨*ProcessDefinition*⟩ '}'

⟨*ProcessDefinition*⟩ ::= ⟨*SBPModel*⟩
| '**`url`**' ':' ⟨*string*⟩

Grammar 5.7: Grammar for defining a SBP state in StratModel

Example 10 *Listing 5.5 presents two states of the transformation cases used to describe the mechanism of the duplication action. We focus on the service 'S2' to show how the states can be described in* StratModel. *The process used to describe the states is our SBP model for molecular evolution reconstruction.*

```
ProcessState state1 {
  Process Process1 {
    ...
        serviceEngine :
          name : 'S2'
          complexity : 'N*N*L + N*L*L'
          groups : c1, c2, c3
          capacity : 600000
          ...
          requests : req1, req2, req3
        ;
        ...
  }
  ...
}
ProcessState state2 {
  Process Process1 {
    ...
        serviceEngine :
          name : 'S2'
          complexity : 'N*N*L + N*L*L'
          groups : c1, c2, c3
          capacity : 600000
          ...
          lb : LB_S2
          copies : S21
          requests : req1, req2, req3
        ;
```

```
        serviceEngine :
          name : 'S21'
          initial : S2
          complexity : 'N*N*L + N*L*L'
          groups : c1
          capacity : 50000
          ...
          lb : LB_S2
        ;
        loadBalancer :
          name : 'LB_S2'
          service : S2
        ;
        ...
  }
  ...
}
...
```

Listing 5.5: Example of describing transformation states with StratModel

5.4.3 StratModel core

After defining an elasticity model using StratModel syntax, the corresponding elasticity controller should be generated from the constructed StratModel document based on a pre-defined template that groups the common functionalities of a controller. In order to achieve this, StratModel language is provided with a set of functionalities that constitute its core. It includes the following components: (1) a validator that allows to check the coherence of the provided elements, (2) a scope provider, and (3) a generator that uses a pre-defined template grouping the common functionalities of a controller to generate an elasticity controller from a given StratModel document.

Usually, a controller is represented by a control loop to perform autonomic management of a system by allowing self-healing, self-protecting, self-configuring and self-optimizing which gives the system the ability to manage its resources automatically and dynamically whenever needed. This loop is named Monitor, Analyze, Plan, Execute, and Knowledge (MAPE-K) loop. It consists in (*i*) harvesting monitoring data, (*ii*) analyzing them using (optionally) a knowledge base and (*iii*) generating reconfiguration actions to correct/prevent violations (self-healing and self-protecting) or to target a new state of the system (self-configuring and self-optimizing) [Jacob 2004].

Figure 5.1 illustrates the pre-defined template used to generate elasticity controllers from elasticity models. It is modelled using high-level petri nets [Jensen 1991] to allow the formal evaluation of elasticity strategies on a SBP model. This template represents the basic construction of elasticity controllers on which the generation is based. It contains a central place of type net system, named BP, representing the managed component and surrounded with a set of transitions for the actions that can be performed on a SBP model (token in the place BP). The *Monitor* transition is used to monitor

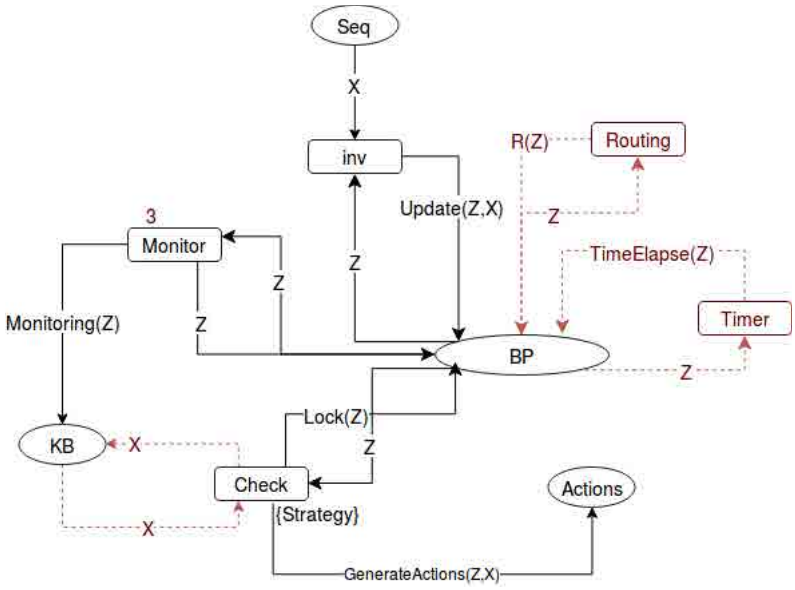


Figure 5.1: Elasticity Controller Petri Net Model Template

the given SBP execution. It is guarded with a delay value representing the frequency of monitoring. For example, if the *Monitor* transition has a value '3' associated to it, it means that there are three cycles between two successive monitoring actions. The firing of this transition adds new value for each registered metric (pre-defined metrics or StratModel metrics) to the place KB that represents the used knowledge base in which the history of monitoring metrics are stored. The *Check* transition is used to inspect a given elasticity strategy and to check for any QoS violations. If it is indicated in the elasticity model that a knowledge base is needed, a connection between the knowledge base component (*i.e.*, place KB) and the *Check* transition will be added in the generated model of elasticity controller allowing the use of the stored information when inspecting the elasticity strategy. The firing of this transition executes a function named $'generateActions$ that generates a set of actions needed to be performed on the SBP model and locks the entities on which the actions will be applied. The place *Actions* holds those actions generated from firing the *Check* transition. The transition *Inv* is used to introduce new requests to the SBP model from the place *Seq* which stores the sequence of requests arrival. According to the elasticity model description, two other transitions can be used (optionally) from the template. The first one is the *Routing* transition which is responsible for transferring requests between services in the SBP model. It can be omitted from the template to allow the user to use a customized routing action defined in his/her elasticity model. The second one is the *Timer* transition which can be used and included in the template when the SBP model includes temporal information. It is used to increment the clocks in the SBP model.

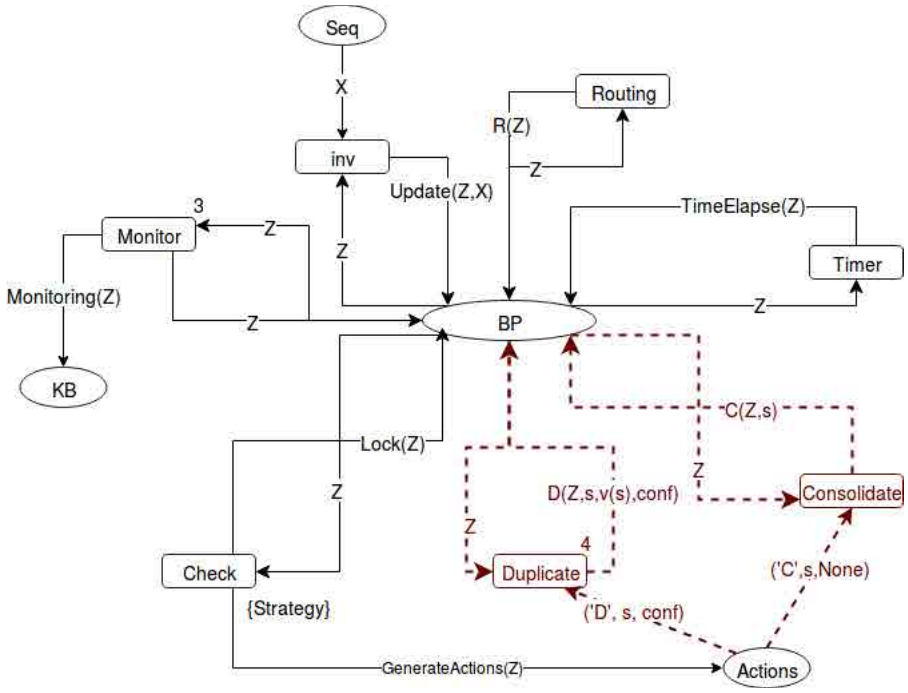


Figure 5.2: Example of a generated elasticity controller Petri Net Model

Given an elasticity model, the elasticity controller petri net model is generated by enriching the pre-defined template with a new set of transitions for the defined actions. Each action is translated to a transition where the transition's name corresponds to the name of the action. It can be associated with a time delay of applying the action. The transition is enabled if there is at least one stored action in the place *Actions* corresponding to the action of the transition. The reference ID of the action is used to identify it in the place *Actions*. The firing of the transition applies the action mechanism on the SBP model using the attributes of the retrieved action from the place *Actions*.

Example 11 *Let's take the elasticity model described in Section 1.3 and specified using* StratModel. *As we previously indicated, it performs two main actions namely Duplicate and Consolidate. So, the default routing mechanism is used by the generated controller. The elasticity strategies that will be used are reactive and they do not need a knowledge base for their elasticity decisions. Figure 5.2 illustrates the generated elasticity controller petri net model. We added two transitions to the template named Duplicate and Consolidate corresponding respectively to the action 'Duplicate' and the action 'Consolidate' in the elasticity model. Since the used SBP model contains temporal information, the Timer transition is allowed in the final controller model. Also, the elasticity model specifies that the default routing mechanism will be used and there is no need for a knowledge base in the analysing step (i.e., the Check transition).*

5.5 STRAT: Elasticity Strategies Description Language

After introducing our StratModel language for describing elasticity model, we present in the following sections our DSL language for defining elasticity strategies for SBPs that relies on elasticity models defined using StratModel.

5.5.1 Strat Overview

Elasticity Strategy Description Language (Strat for short) is a rule-based DSL for specifying strategies governing SBP elasticity. Strat language allows users to specify QoS requirements of a SBP at different granularity levels (*i.e.*, process, service, and instance level) with taking into consideration the fundamental characteristics of elastic SBPs. An elasticity strategy is specified for/based on a specific elasticity model which defines the elasticity capabilities/actions to be used to manage SBPs, the metrics to be called in action's rules and the properties that users can access and reconfigure. The elasticity capabilities are defined using StratModel as objects and then their references are used in Strat language as part of its grammar. Along the elasticity capabilities, StratModel allows users to define a set of customised metrics that can be used as QoS metrics or functions called inside Strat script using their references, and to specify the properties that the users can reconfigure when applying an action according to a Strat strategy. So, a StratModel script is required from the users before defining elasticity strategies to adapt Strat to the described elasticity model.

5.5.2 Strat Grammar

The top-level of Strat specification grammar is given in Grammar 5.8 using the Backus Normal Form (BNF). A Strat document is composed of two main sections encapsulated in a block defined by the keyword `'Strategy'` (*i.e.*, indicates the beginning of the strategy) and identified by a name. The user is allowed to separate the rules section identified by the keyword `'Actions'` from the definition of constants sets used by the rules such as thresholds sets and time constrains. This separation facilitates the maintenance and the adjustment of strategy's code. The latter is an optional section and is identified by keyword `'Sets'`.

The '`Sets`' section as shown in Grammar 5.9 could either be empty or consist of several constants sets. A set can be defined as an upper or lower bound of a quality of service metric by specifying the represented metric, which refers to a metric defined in the provided elasticity model. It allows the specification of the invariant characteristics and requirements of a SBP and its services, *e.g.*, the minimum and the maximum capacity of services in a SBP, time constraint, budget for deployment a service, *etc.* Users can specify these invariants hierarchically from the top-level component, *e.g.*, a process, to its fine-granular level, *e.g.*, a sub-group of service's requests. This is done by using sets of sets which provide the hierarchical construction. So, an item in a set can be associated either to a value or to another set specializing the item. The keyword `'Default'`

⟨*ScalingPolicy*⟩ ::= '**Strategy**' ⟨*name*⟩ '{' ⟨*Statements*⟩ '}'

⟨*Statements*⟩ ::= ⟨*Initialization*⟩ ⟨*ActionsBlock*⟩ | ⟨*ActionsBlock*⟩

⟨*Initialization*⟩ ::= '**Sets**' ':' ⟨*Sets*⟩

⟨*ActionsBlock*⟩ ::= '**Actions**' ':' ⟨*Actions*⟩

Grammar 5.8: General STRAT Grammar

is used in the latter case to provide the requirement (or characteristic) of the item in its top-level. Listing 5.6 illustrates an example of specifying the maximum execution time for two elastic services $s2$ and $s4$ according to their groups of requests $c1$ and $c2$. This specification allows the user to define elasticity action rules that incorporate the characteristic of requests into the decision making.

⟨*Sets*⟩ ::= ⟨*Set*⟩ ';' ⟨*Sets*⟩ | ⟨*empty*⟩

⟨*Set*⟩ ::= ⟨*id*⟩ '=' '{' ⟨*Items*⟩ '}' ⟨*QoS*⟩

⟨*QoS*⟩ ::= '**as**' ⟨*Bound*⟩ [STRATModel::Metric] | ⟨*empty*⟩

⟨*Bound*⟩ ::= '**upper_bound_qos**' | '**lower_bound_qos**'

⟨*Items*⟩ ::= ⟨*Item*⟩ ',' ⟨*Items*⟩ | ⟨*Item*⟩

⟨*Item*⟩ ::= ⟨*id*⟩ '=' ⟨*value*⟩
| ⟨*id*⟩ ':' '{' ⟨*Items*⟩ '}' '`Default`' ⟨*value*⟩

Grammar 5.9: Grammar of defining Sets in STRAT

```
Strategy ...{
Sets:
  max_ex = {s2:{c1=65, c2=120} Default 70,
        s4:{c1=25, c2=75} Default 30}
  ...
Actions:
  ...
}
```

Listing 5.6: Example of defined maximum execution time Set with STRAT

⟨*Actions*⟩ ::= ⟨*Action*⟩ ':' ⟨*Rules*⟩ '.'
| ⟨*Action*⟩ ':' ⟨*Rules*⟩ '.' ⟨*Actions*⟩

⟨*Rules*⟩ ::= ⟨*Rule*⟩ | ⟨*Rule*⟩ ';' ⟨*Rules*⟩

⟨*Rule*⟩ ::= ⟨*Condition*⟩ ⟨*Operator*⟩ ⟨*Condition*⟩
| ⟨*Condition*⟩

⟨*Action*⟩ ::= [StratModel::Action] '(' ⟨*id*⟩ ⟨*Copies*⟩ ⟨*Configuration*⟩ ')' ⟨*Multiple*⟩

⟨*Configuration*⟩ ::= ',' '[' ⟨*ConfigItem*⟩ ⟨*ConfigItems*⟩ ']' | ⟨*empty*⟩

⟨*ConfigItems*⟩ ::= ',' ⟨*ConfigItem*⟩ ⟨*ConfigItems*⟩ | ⟨*empty*⟩

⟨*ConfigItem*⟩ ::= [StratModel::Property] ⟨*ConfigOps*⟩ ⟨*ItemValue*⟩

⟨*Multiple*⟩ ::= '**by**' ⟨*int*⟩ | ⟨*empty*⟩

⟨*Copies*⟩ ::= ',' ⟨*id*⟩ ⟨*Copies*⟩ | ⟨*empty*⟩

⟨*Operator*⟩ ::= '**and**' | '**or**'

⟨*ConfigOps*⟩ ::= '=' | '+=' | '-='

Grammar 5.10: Grammar of specifying Actions and their Rules in Strat

Under the '`Actions`' section, elasticity mechanisms can be provided by setting a set of rules grouped by their intended elasticity action as illustrated in Grammar 5.10. The actions allowed in Strat are the ones specified in the given StratModel script. So, the actions allowed in Strat change by changing the provided script. Moreover, we provide the grammar with a syntactic validator and scope provider modules to adjust actions parameters according to their definitions in the elasticity model by dynamically activate and deactivate parts of syntactic definition of '`Action`'. The parameters of an action are: (1) the elements on which the action will be performed, (2) the new configuration that will be used on the elements when applying the action, and (3) the multiplicity of applying the action. The first two parameters are identified from the defined cases of the action while the multiplicity of the action is allowed if the action is defined as multiple. The configurable items used in the second parameter, *i.e.*, the new configuration, are referred to the properties defined in StratModel as configurable. Each action in Strat can have one or more rules ordered according to their priority. This means that the first provided rule for a specific action is the most priority one from all the action's rules, then the next one is the second most priority and so on. A Strat rule is composed of a set of conditions connected by logical operators (*i.e.*, `and`/`or`).

⟨*Condition*⟩ ::= ⟨*Boolean*⟩ | ⟨*Function*⟩ | ⟨*Comparison*⟩
| ⟨*Iteration*⟩ | ⟨*Condition*⟩ '`for`' ⟨*value*⟩
| '`not`' ⟨*Condition*⟩ | '`(`' ⟨*Rule*⟩ '`)`'

⟨*Comparison*⟩ ::= ⟨*Operand*⟩ ⟨*Ops*⟩ ⟨*Operand*⟩

⟨*Iteration*⟩ ::= '***`foreach`***' ⟨*name*⟩ '***`in`***' ⟨*Sequence*⟩ '`:`' ⟨*Condition*⟩
| '***`exists`***' ⟨*name*⟩ '***`in`***' ⟨*Sequence*⟩ '`:`' ⟨*Condition*⟩

⟨*Sequence*⟩ ::= ⟨*Function*⟩ | '{' ⟨*List*⟩ '}'

⟨*Operand*⟩ ::= ⟨*Function*⟩ | ⟨*value*⟩ | ⟨*SetOperand*⟩ | ⟨*string*⟩

⟨*Function*⟩ ::= ⟨*FunctionName*⟩ '`(`' ⟨*Parameters*⟩ '`)`'

⟨*List*⟩ ::= ⟨*item*⟩ '`,`' ⟨*List*⟩ | ⟨*item*⟩

⟨*FunctionName*⟩ ::= [StratModel::Metric] | '***`STRAT.`***' ⟨*id*⟩

⟨*SetOperand*⟩ ::= [Set] ⟨*QualifiedName*⟩

⟨*QualifiedName*⟩ ::= '`[`' ⟨*item*⟩ '`]`' ⟨*QualifiedName*⟩ | '`[`' ⟨*item*⟩ '`]`'

⟨*Parameters*⟩ ::= ⟨*Operand*⟩ '`,`' ⟨*SetOperands*⟩ | ⟨*Operand*⟩

⟨*item*⟩ ::= ⟨*id*⟩ | ⟨*string*⟩

⟨*Ops*⟩ ::= '`<=`' | '` < `' | '`>=`' | '`>`' | '`==`' | '`!=`'

Grammar 5.11: Grammar for Conditions in Strat

Conditions are the reflection of system's state at a specific point of time. They are split into boolean (*i.e.*, `true`/`false`), boolean function, comparison, iteration, time-based condition, negation, or another rule. The comparison is mathematical comparison between two operands. A operand in Strat can be: (1) a numeric value, (2) a function returning numeric value or a string, (3) a string, or (4) an entry in a set defined in the '`Sets`' section and referred by its name. The time-based condition allows the action execution after some period of time or after the persistence of system's state for some period using the keyword '`for`' to specify that period, *e.g.*, the rule "`Duplicate`(s): true `for` 120 min" schedules a duplication of each elastic service s in every two hours, *i.e.*, "120 min". Unlike existing languages in the literature for describing elasticity strategies, we incorporate the concept of iteration into the definition of Strat to get a global view of the system's state. For example, in order to determine the state of

a service at a given time, we have to check the state of all its copies that will give us a global view instead of a local one (*i.e.*, for only one service's copy). To do so, the keywords `'foreach'` and `'exists'` operators are used to respectively express '∀' and '∃' symbols of the first-order logic. The `Sequence` element of an iteration represents either a function returning a list or a constant list. A function is referred by its name which can be either a reference to a defined StratModel metric or the name of a pre-defined function concatenated to the keyword `'STRAT.'`. The conditions specification in Strat is given by Grammar 5.11 using the Backus Normal Form (BNF).

Example 12 *We describe in Listing 5.7 how the strategy, used in our example in Section 1.3), can be specified using our* Strat *language. We indicate that the set 'max_t' contains the QoS thresholds for each service and group and associated to the metric 'executionTime' defined in the elasticity model 'ElasticityModel1'. The Duplicate action is provided to allow to create a new copy for a specific group with different capacity when the requests of that group with waiting status has been waiting for a certain amount of time. The waiting time thresholds are given in a set named 'max$_w$'.*

```
Strategy Strategy1{
  Sets :
    max_t = {S1:{c1=12, ...} Default 195, ...}
            as upper_bound_qos executionTime
    min_t = {S1 = 1, ...}
          ...
  Actions :
    Duplicate(s,[cat='c3',cap+=600000]) :
      STRAT.has_group(s, 'c3') and
      (exists req in STRAT.waitingRequests(s,'c3') :
        waitingTime(req) >= max_w[s]['c3']) and
      foreach ss in STRAT.copies(s) :
        (exists req2 in STRAT.waitingRequests(ss,'c3') :
          waitingTime(req2) >= max_w[s]['c3']) .
                ...
}
```

Listing 5.7: Describing an elasticity strategy using Strat

5.5.3 Strat Core

It groups the basic functionalities of Strat that allow the processing of Strat scripts, validating the coherence of the given rules and applying them.

- Strat *Scope provider* : The scope provider is responsible for aiding users to edit their script by providing a set of expected elements;
- Strat *Validator* : the validator is responsible for checking rules consistency. We can distinguish two kinds of consistency checking: (1) intra-rule validation that checks contradiction in each rule separately, and (2) inter-rule validation that checks for inconsistencies between rules.

Function	***Description***
services	provides the set of services in the current model (basic+copies)
enabled	checks whether a given router is ready to transfer requests
copies	returns all the copies of a given service

Table 5.1: Examples of STRAT Functions

- STRAT *Generator* : It is a code generator for STRAT language that translates a STRAT script into a GPL such as Python. The generator interacts with the parser used by STRAT to extract the sets and rules from the given script and convert them into GPL (in our case python) code according to a strategy template. The generated code provides mainly the methods used in the generated elasticity controller by STRATModel, namely `'check'` and `'GenerateActions'`. The first one is provided to check for elasticity rules and find whether there is at least one rule applicable for a particular situation while the second method is called by the elasticity controller to generate the applicable actions and their parameters.

- STRAT *Functions* : It provides a set of pre-defined functions used to perform certain tasks on a given business process model. Additionally, it includes the basic functions like arithmetic functions (*i.e.*, add, mul, sub, mod, div), pre-defined metrics and counter function (*i.e.*, count). Some of these functions that can be used in defined a STRAT strategy are given in table 5.1.

5.6 Conclusion

In this chapter, we proposed two linked DSLs used to allow the description of elasticity strategies for different elasticity models. Our first language named STRATModel is designed to define elasticity model for business processes. STRATModel enables business process holders to define their proper elasticity models by describing customized metrics, properties and elasticity actions. The mechanism of a defined action can be provided through a set of examples illustrating how the action should be applying in certain cases. It follows the *by-example* paradigm in generating elasticity mechanisms. Moreover, STRATModel generates customized elasticity controllers. Our Second language named STRAT is designed as an adaptive rule-based DSL for expressing elasticity strategies for SBPs that can be customized according a given elasticity model written using STRATModel. Relying on STRATModel allows STRAT to define elasticity strategies that describe rules for different elasticity actions using customized metrics. Besides its link to STRATModel, STRAT enables the differentiation between requests with different QoS requirements and allows taking several (component) services status when defining elasticity rules.

In the next chapter, we will present the overall framework for evaluating elasticity strategies for business processes in the cloud. This framework basis its evaluation on our formal model for SBPs elasticity and has StratModel and Strat as main building components.

CHAPTER 6

Elasticity Strategies Evaluation Framework

Contents

6.1 Introduction

In the previous chapters, we first introduced our data-aware modeling of SBPs elasticity using high-level petri nets which allows the evaluation of elasticity strategies for SBPs. Then, two languages have been proposed to facilitate the description of elasticity strategies for different elasticity models. We presented a DSL, named StratModel for describing elasticity models and generating their corresponding elasticity controllers using a pre-defined elasticity controller template. We also presented another DSL, named Strat, for describing elasticity strategies for SBPs based on the elasticity capabilities provided in a specific elasticity model written in StratModel.

This chapter presents an evaluation framework, named StratFram, for describing and evaluating elasticity strategies for SBPs using StratModel and Strat languages.

The StratFram framework enables the evaluation, through simulation, of different elasticity strategies for different elasticity models based on our model for elastic execution environment of SBPs. It is designed based on a set of DSLs for describing different simulation elements from the elasticity model to the simulation configuration in order to conceal the used formal methods/systems and the implementation complexity from users. Using StratFram, SBP holders can define (*i*) an elasticity model with specific elasticity capabilities on which they want to define and evaluate their elasticity strategies, (*ii*) a SBP model for which the elasticity strategies will be defined and evaluated, (*iii*) a set of elasticity strategies based on the elasticity model and the provided SBP model, and (*iv*) a simulation configuration which identifies the elements of the evaluation. As results for an evaluation, StratFram provides a set of plots that allows the analysis and the comparison of strategies.

This chapter is organized as follows. In Section 6.2, we introduce our framework architecture. Then, we describe its components, namely, SBP language in Section 6.2.2 and StratSim language in Section 6.2.3. We chain up with the evaluation of our framework in Section 6.3, which contains the implementation details and some experiment results.

6.2 STRATFram: Elasticity Strategies Evaluation Framework for SBPs

Elasticity strategies govern the provisioning of necessary and sufficient resources to ensure the agreed QoS and handle the incoming workload despite variations in enactment requests load. Many strategies can be defined to steer SBPs elasticity. The abundance of possible elasticity strategies requires their evaluation in order to guarantee their effectiveness before using them in real Cloud environments. In this section, we present our elasticity strategies evaluation framework for SBPs, named StratFram.

6.2.1 StratFram overview

Figure 6.1 shows an overview of StratFram framework for evaluating elasticity strategies. StratFram allows users (SBP holders) to evaluate, through simulation, elasticity strategies for a given SBP model and under a given usage behavior based on a specific elasticity model. As illustrated in Figure 6.1, the framework is composed of two main parts: (1) StratFram languages, and (2) StratFram functions. The first part is composed of a set of DSLs including StratModel and Strat (*cf.*, Chapter 5) designed to generalize the use of the framework and to facilitate the description of evaluation elements, *i.e.*, elasticity strategies that will be evaluated, a SBP model for which the elasticity strategies will be defined, an elasticity model on which the elasticity strategies will be based, and simulation configuration that specifies the needed parameters for a simulation, while hiding the implementation complexity and the used formal method from the users. Each language in StratFram is provided with its dedicated editor

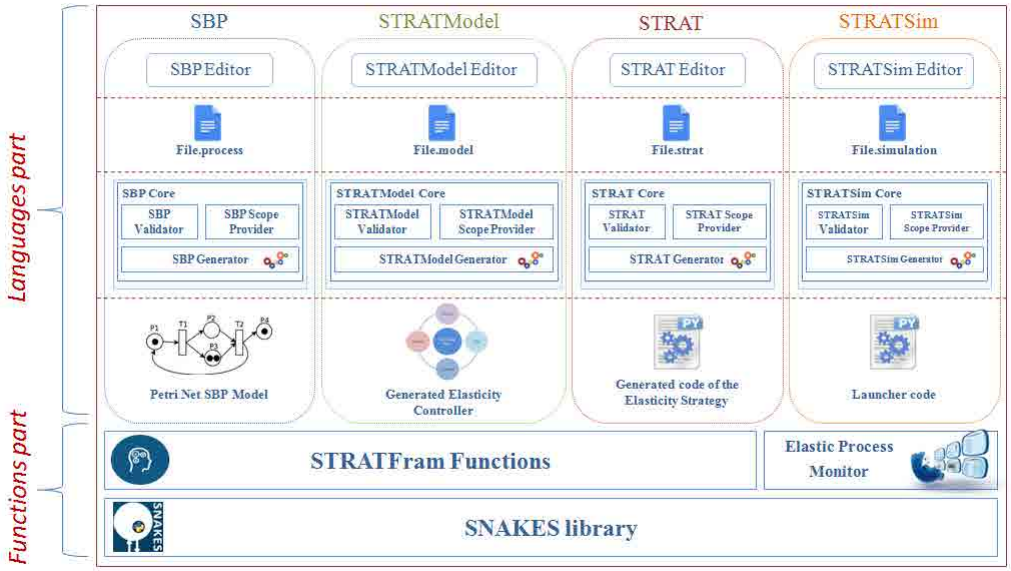


Figure 6.1: StratFram architecture overview

from which the users define the evaluation elements and perform the evaluation. The StratFram languages part is composed of four basic languages: (*i*) SBP language for defining a SBP model that represents the elastic execution environment of a SBP in the cloud, (*ii*) StratModel language presented in Section 5.4, (*iii*) Strat language presented in Section 5.5, and (*iv*) StratSim language for specifying simulation properties/elements in a declarative manner and providing users with a simulation launcher for their evaluation scenario. As we described in the previous chapter, Start language has been designed relying on StartModel language. Other links are also made between StratModel and SBP language, and StratSim and all the other languages. Using StratModel, a user can (optionally) define his elasticity model for a specific SBP by indicating the reference to an already defined SBP model using SBP as the managed component of the elasticity model. Since it is designed to indicate the elements of an evaluation scenario, StratSim provides users with references to the already defined elements that can be used in an evaluation. Each one of these languages is provided with a code generator that is responsible for generating a concrete implementation of the described elements based on the formal method/system used in the second part of the framework. The latter represents the StratFram core/engine that provides common classes and functions used and triggered by the generated items either for processing or profiling the SBP model. The provided classes and functions are implemented based on the underlying used api for a specific formal method/system, *e.g.*, SNAKES API for defining and executing petri-nets.

In the current state of StratFram, the evaluation results are displayed as plots showing the behavior of resource allocations for executing each elastic service in the defined SBP model as well as the one of the overall process according to some defined indicators for a specific elasticity strategy. In the following, we will present our SBP language for describing elastic execution environment of SBPs based on our formal model defined in Chapter 4, followed by the details of StratSim language for describing simulation configurations.

6.2.2 SBP Language

SBP language is proposed to facilitate the description of elastic execution environment of SBPs by using elements composing an elastic execution environment in the cloud. As mentioned in Chapter 4, we argue that it is beneficial to adopt formal models to describe elastic execution environment of SBPs which provides rigorous description and allows formal evaluation and verification of SBPs elasticity. So, we proposed to use petri nets to formally describe SBP models. However, allowing users to express their models in terms of petri nets needs knowledge on petri net concepts and how to use them for describing elastic execution environment of SBPs. To facilitate the modeling of SBPs elasticity and the extensibility of our framework, we propose a DSL, named SBP, for describing an elastic execution environment of SBP and generating its corresponding SBP model according to the provided SBP generator. So, SBP language can be seen as an interface that provides an abstract representation of SBP model entities. By providing different SBP generators for different formal modeling techniques or systems, we allow StratFram to be extensible and used across different systems. In the following, we present the grammar of our SBP language.

6.2.2.1 SBP Grammar

Based on the composition of an elastic execution environment, we define the SBP grammar to allow to describe SBP models using elastic execution environment components rather than using formal notation and terminologies of the used formal method. The top-level of SBP grammar is given in Grammar 6.1 using the Backus Normal Form (BNF).

A SBP document/model is composed of two parts: (1) a structure description part and (2) a marking part that corresponds to a set of requests in the process. The first part is specified in a block defined by the keyword `'Process'` to indicate the beginning of the SBP structure description and identified by a name. As described in the formal model presented in Chapter 4, the structure of a SBP can be defined by four sets: (*i*) a set of groups, (*ii*) a set of nodes where a node is either a service engine or a load balancer, (*iii*) a set of routers, and (*iv*) a set of directed links connecting either nodes to routers or routers to nodes. To describe a SBP, the user/SBP holder can provide the process with a reference ID to indicate its name outside the framework and with a descriptive text.

⟨*SBPModel*⟩ ::= '`Process`' ⟨*id*⟩ '{' ⟨*ProcessDescription*⟩ '}' ⟨*Requests*⟩

⟨*ProcessDescription*⟩ ::= ⟨*Reference*⟩ ⟨*Description*⟩ ⟨*Groups*⟩ ⟨*Nodes*⟩ ⟨*Routers*⟩ ⟨*Links*⟩

⟨*Groups*⟩ ::= ⟨*Group*⟩ ⟨*Groups*⟩ | ⟨*empty*⟩

⟨*Nodes*⟩ ::= ⟨*Node*⟩ ⟨*Nodes*⟩ | ⟨*Node*⟩

⟨*Node*⟩ ::= ⟨*ServiceEngine*⟩ | ⟨*LoadBalancer*⟩

⟨*Routers*⟩ ::= ⟨*Router*⟩ ⟨*Routers*⟩ | ⟨*empty*⟩

⟨*Links*⟩ ::= ⟨*Link*⟩ ⟨*Links*⟩ | ⟨*empty*⟩

⟨*Requests*⟩ ::= ⟨*Request*⟩ ⟨*Requests*⟩ | ⟨*empty*⟩

Grammar 6.1: General SBP Grammar

A group represents a set of requests with specific characteristics/requirements. As illustrated in Grammar 6.2, it is described by a name which refers to a tenant, a group of users (such as premium user, normal user, and guest user), or a group of requests (*i.e.*, `REQUEST_DATA`) according to the indicated type. The group of type `REQUEST_DATA` is defined depending on the size of requests data. In order to specify the boundary of such a group, the user is allowed in SBP to indicate the range of requests data size by specifying the maximum size for a request to be considered as a member of the group. This categorization of requests allows users (SBP holders) to specify different QoS requirements for different groups and assign groups to specific service engine copies.

A service engine represents the container on which a service is hosted. A container can be either a VM or a container like Docker [Merkel 2014] or micro-container [Yangui 2011]. A service engine, as shown in Grammar 6.3, is described by a set of attributes specifying its characteristics. A name is given to a service engine to identify it inside the framework while its reference indicates its actual name or location. A service engine can be created for a particular set of groups. So, users can associate a set of groups to the service engine by referring to their names as given in the groups description section. For the simulation purpose, a complexity function can be provided for a service engine to refer to the time complexity of the hosted service and therefore to estimate the processing time needed for handling each request. The capacity and processing speed attributes can be given to indicate the quantity of data that can be processed simultaneously by the service engine and how fast this processing can be done. These processing specifications can be provided to a service with a cost which may restrict the use of resources. The timeout attribute can be given to indicate after

---

⟨*Group*⟩ ::= '**group**' ':' ⟨*GroupDescription*⟩ ';'

⟨*GroupDescription*⟩ ::= ⟨*Name*⟩ ⟨*Type*⟩ ⟨*Range*⟩

⟨*Name*⟩ ::= '**name**' ':' ⟨*id*⟩

⟨*Type*⟩ ::= '**type** ':' ⟨*GroupType*⟩ | ⟨*empty*⟩

⟨*Range*⟩ ::= '**range**' ':' ⟨*value*⟩ | ⟨*empty*⟩

⟨*GroupType*⟩ ::= 'REQUEST_DATA' | 'REQUEST_USER' | 'TENANT'

Grammar 6.2: Grammar for describing process groups in SBP

how much time a request should become (be considered) outdated inside the service engine. Additionally, a service engine can be described as an elastic service engine or not. In the first case, it may have many copies hosting the same service and connected to a particular load balancer. So, '*initial*', '*copies*' and '*lb*' attributes are used to refer respectively to the original service engine copy from which the current one has been duplicated, the list of service engine copies which are given by referring to their names, and a reference to the load balancer node that is associated to the current service. Moreover, the set of belonging requests can be given by referring to their identifiers in the request description section.

After introducing the service engines composing a SBP, load balancers can be provided for the described elastic service engines to connect theirs copies and balance the incoming load between them. As shown in Grammar 6.4, a load balancer in SBP is primary described by a name, (optionally) a reference and an associated service engine. It represents a service that is responsible for receiving requests and forward them to a specific service engine copy in order to balancer the load between different copies of the same service according to a load balancing algorithm such as Round Robin. Additionally, a load balancer may have a capacity (or a queue) representing the maximum number of requests that can keep in its queue before forward them to their target service engine copy. Similar to service engines, it can also have a temporal information (*i.e.*, timeout) indicating when a request in the queue will be considered outdated, a set of allowed groups referred to by their name as provided in the groups description section, and a set of belonging requests.

In order to transfer requests between services in the SBP (either service engines or load balancers), a router is used to connect them to each other and to take requests from services as output and transfer them to the next services as input. For simulation purpose, a router can be provided with an expression representing a condition on requests to be transferred by it. As we may have more than one copy of a service engine, we can

⟨*ServiceEngine*⟩ ::= '`serviceEngine`' ':' ⟨*SEDesc*⟩ ';'

⟨*SEDesc*⟩ ::= ⟨*Name*⟩ ⟨*Reference*⟩ ⟨*OriginalCopy*⟩ ⟨*Container*⟩ ⟨*Complexity*⟩ ⟨*NodeGroups*⟩ ⟨*Capacity*⟩ ⟨*Speed*⟩ ⟨*Elastic*⟩ ⟨*Cost*⟩ ⟨*TimeOut*⟩ ⟨*ServiceLoadBalancer*⟩ ⟨*ServiceCopies*⟩ ⟨*NodeRequests*⟩

⟨*Name*⟩ ::= '`name`' ':' ⟨*string*⟩

⟨*Reference*⟩ ::= '`referenceID`' ':' ⟨*id*⟩ | ⟨*empty*⟩

⟨*OriginalCopy*⟩ ::= '`initial`' ':' [ServiceEngine] | ⟨*empty*⟩

⟨*Container*⟩ ::= '`container`' ':' ⟨*ContainerType*⟩ | ⟨*empty*⟩

⟨*Complexity*⟩ ::= '`complexity`' ':' ⟨*string*⟩ | ⟨*empty*⟩

⟨*NodeGroups*⟩ ::= '`groups`' ':' ⟨*SetGroups*⟩ | ⟨*empty*⟩

⟨*SetGroups*⟩ ::= [Group] ',' ⟨*SetGroups*⟩ | [Group]

⟨*Capacity*⟩ ::= '`capacity`' ':' ⟨*Double*⟩ ⟨*CapacityUnit*⟩ | ⟨*empty*⟩

⟨*Speed*⟩ ::= '`speed`' ':' ⟨*Double*⟩ ⟨*SpeedUnit*⟩ | ⟨*empty*⟩

⟨*elastic*⟩ ::= '`elastic`' ':' ⟨*Boolean*⟩ | ⟨*empty*⟩

⟨*Cost*⟩ ::= '`cost`' ':' ⟨*Double*⟩ ⟨*CostUnit*⟩ | ⟨*empty*⟩

⟨*TimeOut*⟩ ::= '`timeout`' ':' '`after` ⟨*Double*⟩ ⟨*TimeUnit*⟩ | ⟨*empty*⟩

⟨*ServiceLoadBalancer*⟩ ::= '`lb`' ':' [LoadBalancer] | ⟨*empty*⟩

⟨*ServiceCopies*⟩ ::= '`copies`' ':' ⟨*SetCopies*⟩ | ⟨*empty*⟩

⟨*SetCopies*⟩ ::= [ServiceEngine] ',' ⟨*SetCopies*⟩ | [ServiceEngine]

⟨*NodeRequests*⟩ ::= '`requests`' ':' ⟨*SetRequests*⟩ | ⟨*empty*⟩

⟨*SetRequests*⟩ ::= [Request] ',' ⟨*SetRequest*⟩ | [Request]

Grammar 6.3: Grammar for describing ServiceEngine in SBP

⟨*LoadBalancer*⟩ ::= '**loadBalancer**' ':' ⟨*LBDesc*⟩ ';'

⟨*LBDesc*⟩ ::= ⟨*Name*⟩ ⟨*Reference*⟩ ⟨*Service*⟩ ⟨*NodeGroups*⟩ ⟨*Queue*⟩ ⟨*TimeOut*⟩ ⟨*Algorithm*⟩ ⟨*NodeRequests*⟩

⟨*Name*⟩ ::= '**name**' ':' ⟨*string*⟩

⟨*Reference*⟩ ::= '**referenceID**' ':' ⟨*id*⟩ | ⟨*empty*⟩

⟨*Service*⟩ ::= '**service**' ':' [ServiceEngine]

⟨*NodeGroups*⟩ ::= '**groups**' ':' ⟨*SetGroups*⟩ | ⟨*empty*⟩

⟨*SetGroups*⟩ ::= [Group] ',' ⟨*SetGroups*⟩ | [Group]

⟨*Queue*⟩ ::= '**queue**' ':' ⟨*Double*⟩ | ⟨*empty*⟩

⟨*TimeOut*⟩ ::= '**timeout**' ':' ⟨*Double*⟩ ⟨*TimeUnit*⟩ | ⟨*empty*⟩

⟨*Algorithm*⟩ ::= '**algorithm**' ':' ⟨*LBAlgorithm*⟩ | ⟨*empty*⟩

⟨*NodeRequests*⟩ ::= '**requests**' ':' ⟨*SetRequests*⟩ | ⟨*empty*⟩

⟨*SetRequests*⟩ ::= [Request] ',' ⟨*SetRequest*⟩ | [Request]

Grammar 6.4: Grammar for describing LoadBalancers in SBP

also have a set of routers related by equivalent relation connected to the service engine copies. So, in the router description, the user can identify the initial router element, from which the other copies are copied, by referring to its name. Additionally, the router copies can be specified by referring to their name as a list. The formal grammar for defining routers is given in Grammar 6.5.

Therefore, for routers to be able to do their function, they have to be connected to services via input and output links. As illustrated in Grammar 6.6, an input link connects a router to a service (either a service engine or a load balancer) and gives requests to the latter as input. On the other hand, an output link connects a service to a router and takes requests from the former as output. In order to describe the changes that can be made by services on input requests, the user can provide links with an expression that allows to modify the characteristics of requests when they are transferred from node to node.

A request can be characterized in SBP by a set of attributes defining its static and dynamic aspects. The static characteristics of a request are its identifier and its

⟨*Router*⟩ ::= '**router**' ':' ⟨*RouterDesc*⟩ ';'

⟨*RouterDesc*⟩ ::= ⟨*Name*⟩ ⟨*OriginalRouter*⟩ ⟨*Expression*⟩ ⟨*RouterCopies*⟩

⟨*Name*⟩ ::= '**name**' ':' ⟨*string*⟩

⟨*OriginalRouter*⟩ ::= '**initial**' ':' [Router] | ⟨*empty*⟩

⟨*Expression*⟩ ::= '**guard**' ':' ⟨*string*⟩ | ⟨*empty*⟩ ⟨*RouterCopies*⟩ ::= '**copies**' ':' ⟨*SetRouters*⟩ | ⟨*empty*⟩

⟨*SetRouters*⟩ ::= [Router] ',' ⟨*SetRouter*⟩ | [Router]

Grammar 6.5: Grammar for describing process routers in SBP

⟨*Link*⟩ ::= ⟨*InLink*⟩ | ⟨*OutLink*⟩

⟨*InLink*⟩ ::= '**in**' ':' [Router] '**to**' [Node] ⟨*LinkExpression*⟩

⟨*OutLink*⟩ ::= '**out**' ':' [Node] '**to**' [Router] ⟨*LinkExpression*⟩

⟨*LinkExpression*⟩ ::= '**with**' ⟨*string*⟩ | ⟨*empty*⟩

Grammar 6.6: Grammar for specifying links between nodes and routers in SBP

reference. The same request sent by a user traverses a set of services to complete the final result. So, some request's characteristics can be changed when transferred from service to service, such as its data size and its belonging group (if the group type is data based) which can take new values according to the output data of one service and the input data of the next service. The request data size can be provided as a set of tuples indicating the size of different dimension of the input data. For example, if a request input data is a two dimensional matrix with 'x' lines and 'y' columns, the user can specify using SBP its size as a tuple (x,y). Also, a request may belong to a specific group referred by its name as defined in the groups description section and exists a specific node, either a service engine node or a load balancer node specified by its name. Some other request characteristics/attributes are changeable inside the node such as the request's age which represents the processing/waiting timeand the request's status which defines the request's progress (waiting/running/finished/dead).

As shown in Grammar 6.7, more than one attribute can be specified for a request to indicate time information. The '*age*' attribute can be provided to indicate the total wait-

ing and processing time of the request since it entered the service. The '*executionTime*' attribute represents the processing time of a request and from which the waiting time can be concluded while the '*totalTime*' attribute represents the total time spent by the request since the beginning of the process.

⟨*Request*⟩ ::= '`Request`' ⟨*id*⟩ '{' ⟨*RequestDesc*⟩ '}'

⟨*RequestDesc*⟩ ::= ⟨*Reference*⟩ ⟨*RequestNode*⟩ ⟨*RequestGroup*⟩ ⟨*RequestStatus*⟩ ⟨*RequestAge*⟩ ⟨*RequestExecutionTime*⟩ ⟨*RequestTotalTime*⟩ ⟨*RequestData*⟩

⟨*Reference*⟩ ::= '`referenceID`' ':' ⟨*id*⟩ | ⟨*empty*⟩

⟨*RequestNode*⟩ ::= '`node`' ':' [Node]

⟨*RequestGroup*⟩ ::= '`group`' ':' [Group] | ⟨*empty*⟩

⟨*RequestStatus*⟩ ::= '`status`' ':' ⟨*Status*⟩ | ⟨*empty*⟩

⟨*RequestAge*⟩ ::= '`age`' ':' ⟨*Double*⟩ ⟨*TimeUnit*⟩ | ⟨*empty*⟩

⟨*RequestExecutionTime*⟩ ::= '`executionTime`' ':' ⟨*Double*⟩ ⟨*TimeUnit*⟩ | ⟨*empty*⟩

⟨*RequestTotalTime*⟩ ::= '`totalTime`' ':' ⟨*Double*⟩ ⟨*TimeUnit*⟩ | ⟨*empty*⟩

⟨*RequestData*⟩ ::= '`data`' ':' ⟨*Tuple*⟩ | ⟨*empty*⟩

⟨*Tuple*⟩ ::= ⟨*Double*⟩ ',' ⟨*Tuple*⟩ | ⟨*Double*⟩

⟨*Status*⟩ ::= '`WAITING`' | '`RUNNING`' | '`FINISHED`' | '`DEAD`'

Grammar 6.7: Grammar for defining Requests in SBP

Example 13 *Listing 6.1 describes our SBP model, presented in Section 1.3, using SBP language. It presents the description of a group named 'c1' that represents a group of small requests. The group is defined depending on the size of requests data. In our motivating example, the requests data size is the size of a two-dimension matrix, cf., $N \times M$ where N is the number of lines in the input data and M is the number of column (i.e., the length of a sequence). The group 'c1' is one of the groups associated to the service engine 'S2'. This service engine is of time complexity $O(N^2 \times L + N \times L^2)$. It has a capacity of $6{\times}10^5$ (maximum quantity of request data that the service engine can simultaneously process) and is allowed to process 10^5 instructions per time unit. Also, the service engine 'S2' considers requests that has been in it for more than 2440 time*

units as an outdated request. Initially, 'S2' does not need a load balancer service to balance the incoming load. With the creation of more new service copies to serve the increasing incoming load, a load balancer service will be integrated in the SBP model and used as an entry point to the copies of 'S2'. So, a load balancer service named 'B2' is specified for 'S2' and characterized with a queue indicating how many requests it can keep waiting before forwarding them to a specific copy of the service. The requests in 'B2' are considered outdated after 150 time units of waiting in the queue.

```
Process MERProcess{
  referenceID : 'MolecularEvolutionReconstruction'
  group :
        name : c1
        type : REQUEST_DATA
        range : 10000
  ;
    ...
  serviceEngine :
        name : 'S2'
        complexity : 'N*N*L+N*L*L'
        groups : c1, c2, c3
        capacity : 600000
        speed : 200000
        elastic : true
        timeout : 2440
  ;
  loadBalancer :
        name : 'B2'
        service : S2
        groups : c1, c2, c3
        queue: 10
        timeout : 150
  ;
    ...
  router : name : 'T1'
        ...
  out : S1 to R1
  in : R1 to S2
        ...
}
Request Req1 {
        referenceID : 'req1'
        node : S2
        group : c1
        age : 0
        executionTime : 0
        totalTime : 0
        data : 100.0,100.0
}
```

Listing 6.1: Example of describing an SBP Model with SBP language

6.2.2.2 SBP Core

After describing a SBP model using SBP language, the resulting script have to be processed to generate it corresponding formal model according to the chosen technique on which the evaluation of elasticity strategies will be performed. In order to do so, we provide the language with some basic functionalities constituting its core/engine. This part is composed of the three main components that can be adapted to the chosen formal technique by providing a new implementation.

- SBP *Validator* : The validator is responsible for checking the coherence of the described model structure and the correctness of the expressions provided for routers and links. The checking of correctness is performed according to the syntax of expressions as provided by the used tool of formal modeling technique such as SNAKES toolkit [Pommereau 2015] for petri nets.

- SBP *Scope provider* : The scope provider is responsible for aiding users to edit their SBP scripts by providing them with a set of expected elements;

- SBP *Generator* : The code generator in SBP language translates a SBP document into the corresponding formal model. In the current version of StratFram, we provide SBP language with a code generator that allows to generate a petri net model from a SBP document/script. According to the described SBP model, the generator chooses to generate: (*i*) a stateless petri net model where the requests/tokens are defined as black tokens, (*ii*) a timed petri net model where a request is specified with time information, or (*iii*) a timed colored petri net model, as presented in Chapter 4, that presents a request with certain information such as a belonging group, time information, and data.

6.2.3 StratSim language

StratSim is a DSL for specifying simulation properties/elements in a declarative manner and generating a simulator launcher that will trigger the evaluation of the specified elasticity strategies. StratSim document is composed of a set of (property, value) pairs representing the parameters that customize the simulator to perform the wanted simulations.

As shown in Grammar 6.8, users have to specify the elasticity model on which the simulations will be based on, by referring to an existing elasticity model (described using StratModel). Also, they can provide a set of items composed of: (*i*) processes described using SBP language, (*ii*) elasticity strategies written in Strat language, and (*iii*) workloads representing different configurations of specifying the arrival law of process enactment requests. These items indicate the series of simulations that have to be performed by the simulator. Additionally, the users can optionally indicate the process invocations frequency which represents the distance between two process

⟨*Simulation*⟩ ::= ⟨*ElasticityModel*⟩ ⟨*Items*⟩ ⟨*Invocation*⟩ ⟨*OutputPath*⟩

⟨*ElasticityModel*⟩ ::= '`elasticityModel`' '`:`' [StratModel::ElasticityModel]

⟨*Items*⟩ ::= ⟨*Item*⟩ ⟨*Items*⟩ | ⟨*empty*⟩

⟨*Item*⟩ ::= ⟨*Process*⟩ | ⟨*Strategy*⟩ | ⟨*UsageBehavior*⟩

⟨*Process*⟩ ::= '`process`' '`:`' [SBP::SBPModel]

⟨*Strategy*⟩ ::= '`strategy`' '`:`' [Strat::ScalingPolicy]

⟨*UsageBehavior*⟩ ::= '`workload`' '`:`' ⟨*string*⟩ ⟨*Values*⟩

⟨*Invocation*⟩ ::= '`frequency`' '`:`' ⟨*int*⟩
| ⟨*empty*⟩

⟨*OutputPath*⟩ ::= '`output`' '`:`' ⟨*string*⟩

Grammar 6.8: General StratSim Grammar

invocations. Finally, a property named '*output*' is used to specify the path in which the evaluation results will be saved.

Example 14 *Listing 6.2 illustrates an example of simulation script writen in* Strat*Sim. This script indicates that the elasticity strategy 'Strategy*1*' (presented in Section 5.5) will be evaluated on the SBP model 'MERProcess' using the elasticity model 'ElasticityModel*1*' (presented in Section 5.4). The poisson distribution will be used to generate the requests arrival and a number of requests will be introduced to the process each 12 cycles.*

elasticityModel *: ElasticityModel1*
process *: MERProcess*
strategy *: Strategy1*
workload *: '...generators.hpoisson' 4 100*
frequency *: 12*
output *: 'path'*

Listing 6.2: Example of simulation scenario with StratSim

6.3 Implementation and evaluation

We present in this section an overview of the implementation of StratFram framework. We also provide insights on its use through two evaluation scenarios.

6.3.1 Implementation

The StratFram frameworkis an eclipse-based framework. It is designed and implemented on two parts: (1) the StratFram languages part and (2) the StratFram functions part.

The first part is developed using Xtext[1] an eclipse-based development framework for creating DSLs. Each designed language in StratFram has its editor which provides user with code completion, syntax highlighting, automated parsing and quick fixes functionalities that facilities the edition of scripts. In addition to these functionalities, we provided each language with a scope provider that is customized according to the semantics of the language, along with a validator that is implemented to perform additional constraint checks.

In the current implementation of StratFram, the second part of the framework is composed of a set of functions and classes designed and implemented based on SNAKES toolkit [Pommereau 2015] a Python library that provides all the necessary to define and execute many sorts of Petri nets, in particular algebras of Petri nets. The functions and classes implemented in this part are mainly provide the common functionalities of StratFram that manipulate SBP models and triggered by the generated elasticity controller. A set of functions are implemented as a pre-defined functions for Strat language that can be used in defined a strategy.

6.3.2 Evaluation

In order to validate our framework, we present in this section two evaluation scenarios: the first one focuses on simulating the use of a strategy on publicly available SBPs using an elasticity model, while the second one consists on two use cases for two different elasticity model and using our SBP model for molecular evolution construction in which different elasticity strategies are evaluated and compared. Thereafter, we describe the preliminary results.

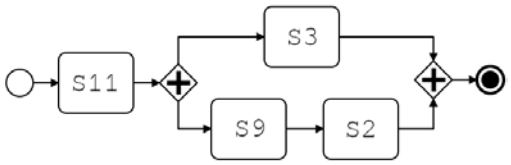
6.3.2.1 1^{st} Evaluation Scenario

In this evaluation scenario, we choose to evaluate the elasticity of a set of publicly available process models selected from the SAP reference model [Cur 1998], using an elasticity model described according to the elasticity controller used in [Amziani 2013]. The SAP reference model has been widely used in many research papers [Mendling 2008]. It contains 205 business process models. A node in a process model can be a function/service, an event or a gateway. In order to use the available process models in SAP reference model, we transformed the models to a SBP models by eliminating the events and their connections and focusing on services/functions. From the transformed models, we selected a set of 5 exemplary SBP models which feature different complexity degrees in terms of business process patterns. Table 6.1 shows the characteristics of each

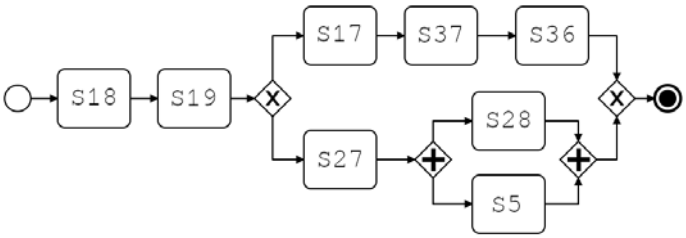
[1] https://eclipse.org/Xtext/

SBP	$\|Services\|$	$\|AND\|$	$\|XOR\|$
1	3	0	0
2	3	0	1
3	4	1	0
4	5	1	3
5	8	1	1

Table 6.1: Evaluation SBP models



(a)



(b)

Figure 6.2: BPMN models of (a) the SBP No. 3 (top) and (b) the SBP No. 5

selected SBP models in term of number of services in the SBP, number of AND-blocks and number of XOR-blocks. Figure 6.2 illustrates two SBP models from the selected set of SBPs. The first model ,named *SBP No. 3*, contains 4 services and one AND-block. The second model, named *SBP No. 5* contains 8 services and one XOR-block in which one branch leads to an AND-block.

For each service in the SBP models, we associate a capacity value randomly generated that indicates the maximum number of requests that can be handled by the service. So, in order to evaluate the elasticity of the SBP models, we choose to describe an elasticity model for the elasticity controller proposed in [Amziani 2013] which has been proposed for stateless SBP models and performs the following elasticity actions: (1) a *Routing* action which controls the way a load of a service is routed over the set of its copies, (2) a *Duplicate* action which creates a new exact copy of an overloaded service in order to meet its workload increase and (3) a *Consolidate* action which releases an unnecessary copy of a service in order to meet its workload decrease. Along with these actions, we specify a '*capacityUtilization*' metric that provides the number of requests in a given service. In this evaluation scenario, we choose to define elasticity strategies

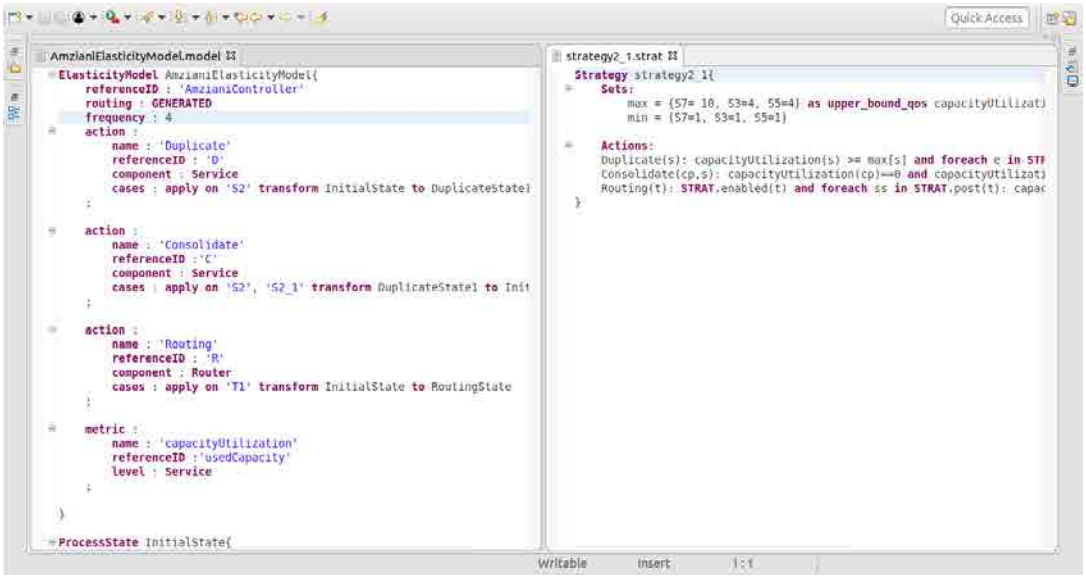


Figure 6.3: Elasticity model and strategy of the 1^{st} Evaluation Scenario

for our SBP models that share the same set of rules while changing the defined threshold sets which have been specified according to the generated capacity values.

The strategy is defined to scale up or down the number of copies according to the threshold of each service. So, the duplication action for a specific service s is triggered when its workload (determined by the '*capacityUtilization*' metric) as well as the workload of all its copies reached its QoS threshold in the set max represented by the metric '*capacityUtilization*'. Otherwise, in case a service copy cp doesn't contain any request and the workload of the service s is below its minimum threshold defined in the set min, a consolidation action is triggered by releasing the service copy cp from the set of copies of s. The Routing action is defined to route a request if the router t does not exceed the capacity of its post-services. It is triggered when neither of the previous actions are allowed. Figure 6.3 shows the described elasticity model and the used elasticity strategy.

By this scenario, we show how the same elasticity model and strategy can be reused with different SBP models without making any changes as long as the logic and the requirements are the same. The only change made in the strategy is the values in the defined sets which depend on the services in the managed process.

6.3.2.2 2^{nd} Evaluation Scenario

Using StratFram interface, we create two projects where in each one we create an elasticity model using StratModel editor, a SBP model using SBP editor, three elasticity strategies using Strat editor, and a simulator script using StratSim editor. In

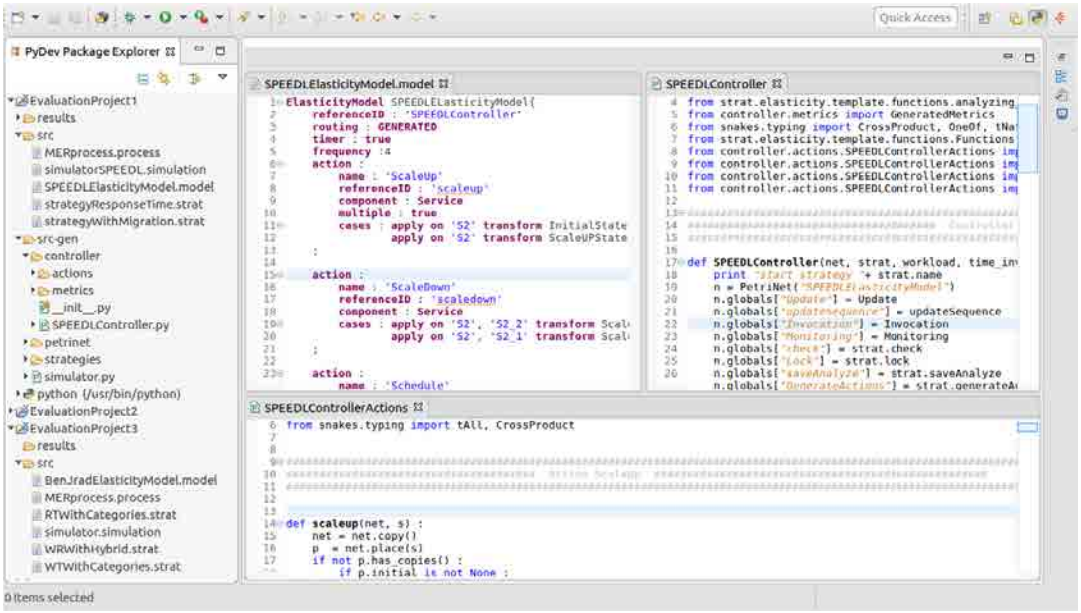


Figure 6.4: SPEEDL elasticity model and its generated controller and actions

the both projects, we use the SBP model for molecular evolution reconstruction (MER) presented in Section 1.3. We assume that each elastic service engine is provided with a maximum response time thresholds (execution time) depending on requests groups as their required QoS given with the elasticity strategies. Above the maximum threshold, the QoS would no longer be guaranteed. We used the following QoS requirement in each elasticity strategy defined in the projects:

- Max_t = {S1:{c1=12,c2=102,c3=202},
  S2:{c1=10,c2=500,c3=1995},
  S5:{c1=4,c2=255,c3=1005},
  S6.1:{c1=4,c2=130,c3=1005},
  S6.2:{c1=4,c2=130,c3=1005},
  S7:{c1=100,c2=1005,c3=2005}}

1. ***Evaluation project 1:*** Along with the SBP model, this project is composed of the following elements :

   (a) The elasticity model on which SPEEDL language [Zabolotnyi 2015] is based. It is composed of four main actions: (1) Request scheduling that represents the mapping of a request to the next service engine in the process, (2) Request migration which re-maps an already existing request to another service engine copy, (3) scale-up that adds a new service engine copy, and (4) scale-down that remove unused service engine copy. With the set of actions, we

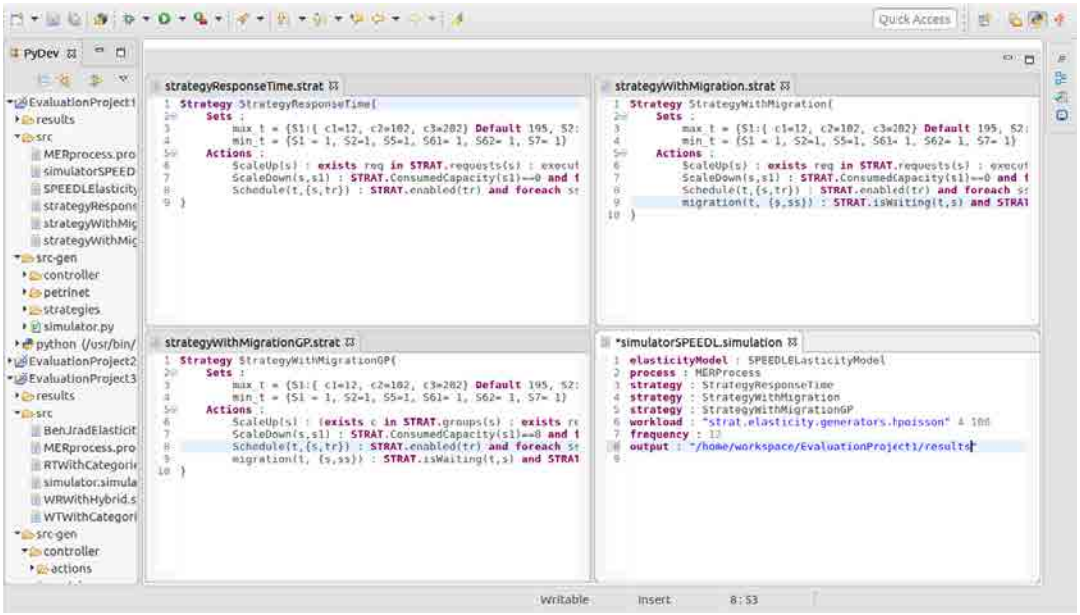


Figure 6.5: Elasticity strategies and simulation script for Evaluation project 1

define in this elasticity model '*executionTime*' metric that provides the age (execution time) of a given request. Figure 6.4 illustrates the elasticity model script and its generated controller and actions implementation.

(b) A strategy named `StrategyResponseTime` that defines rules for scaling-up, scaling-down, and scheduling of request. In this strategy, the conditions of elasticity actions do not consider the distinguishing between services requests and their different QoS requirements. So, we define a fixed maximum response time threshold for each elastic service engine regardless of requests groups as an elasticity indicator. The scale-up action is triggered when the response time of at least one service request in each service engine copy has reached the maximum response time threshold. Otherwise, if the consumed capacity of a service engine copy is equal to 0 and the response time of the service is below its minimum threshold, a scale-down action is triggered by releasing the service engine copy.

(c) A strategy named `StrategyWithMigration` that defines rules for scaling-up, scaling-down, scheduling of requests, and migrating of requests. In this strategy as well, the conditions of elasticity actions do not consider the distinguishing between services requests and their different QoS requirements. So, we use the same defined fixed maximum response time threshold. This strategies provides a rule for migration action in addition to the rules defined in the strategy `StrategyResponseTime`.

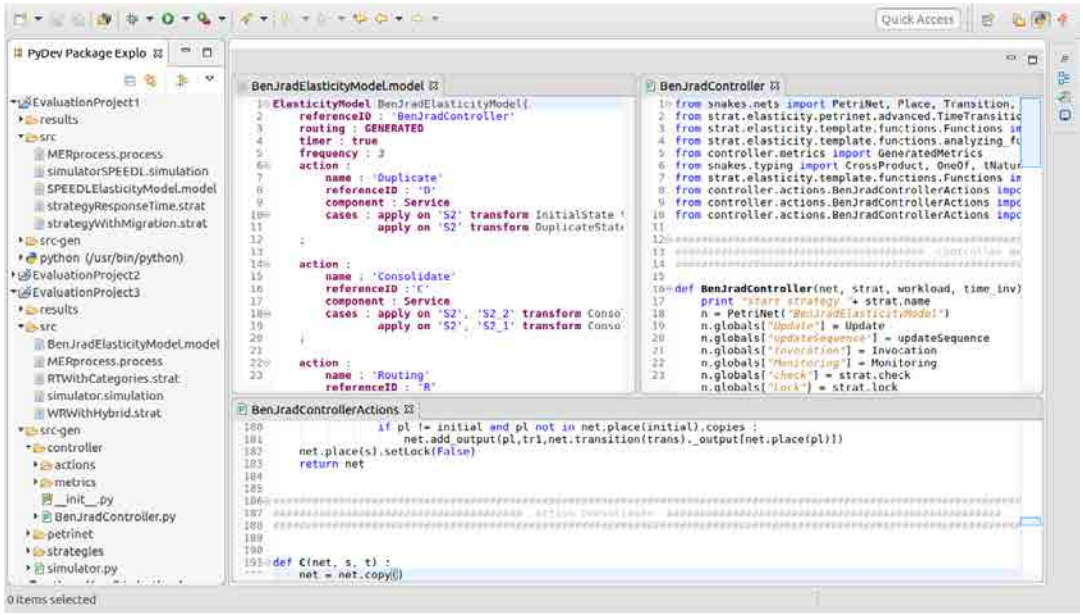


Figure 6.6: Jrad *et al.* elasticity model and its generated controller and actions

(d) A strategy named `StrategyWithMigrationGP` that defines rules for scaling-up, scaling-down, scheduling of requests, and migrating of requests. Contrary to the previous strategies, this strategy uses the provided QoS requirement as an elasticity indicator to scale-up the service engine while uses the same rules as in the strategy `StrategyWithMigration` for the other actions.

(e) A simulation script that specifies that the three provided strategies should be evaluated and compared on the SBP model using the elasticity controller generated from the SPEEDL elasticity model. Figure 6.5 illustrates the defined strategies and the simulation script.

2. ***Evaluation project 2:*** Along with the SBP model, this project is composed of the following elements:

(a) An elasticity model for hybrid scaling (*cf.*, Section 4.4) is defined composed of three main actions: (1) duplication action that adds a new copy of service engine with different configuration (2) consolidation action that removes unused service engine copy, (3) routing action which transfers request from a service engine to the next service engine in the process. With the set of actions, we define in this elasticity model '*executionTime*' metric and '*waitingTime*' metric that provide respectively the age (execution time) and the waiting time of a given request. Also, we specify two properties as re-configurable namely the property 'groups' and the property 'capacity'. Figure 6.6 illustrates the elasticity model script and its generated controller

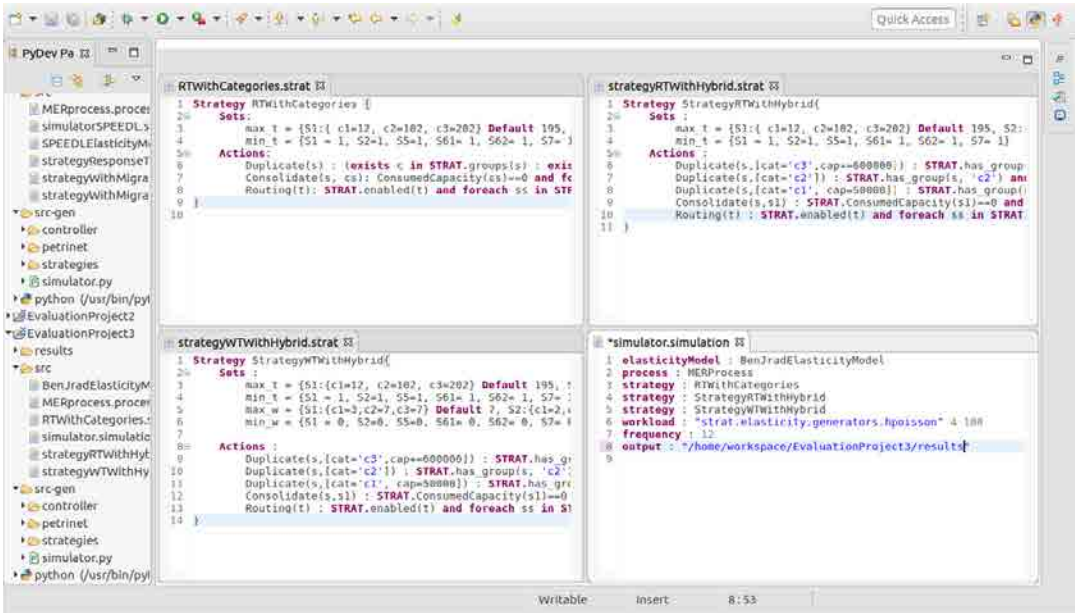


Figure 6.7: Elasticity strategies and simulation script for Evaluation project 2

and actions implementation.

(b) A strategy named `RTWithCategories` that defines rules for duplicate action, consolidate action, routing action. It uses the provided QoS requirement as an elasticity indicator to duplicate the service engine and the minimum response time thresholds an indicator to consolidate a service engine copy. So, the duplication action for a specific service engine s is triggered when the maximum response time threshold has been reached for at least one of request groups and the same applied for all its copies. Otherwise, if the consumed capacity of a service engine copy is equal to 0 and the response time of the service is below its minimum threshold, a consolidate action is triggered by releasing the service engine copy.

(c) A strategy named `StrategyRTWithHybrid` that defines rules for hybrid scaling that allows to add a new service engine copy with specific configuration whenever needed. So, the duplication action for a specific service engine is triggered depending on the requests groups. It creates a new copy of a service engine for requests under a specific group if at least one of requests of that group exceeds the maximum response time threshold and the same applied to all its copies. The consolidation and routing rules are the same for all the strategies defined in this project.

(d) A strategy named `StrategyWTWithHybrid` that defines rules for hybrid scaling that allows to add a new service engine copy with specific configuration

whenever needed based on the waiting time of requests. So, the duplication action creates a new copy of a service engine for requests under a specific group if at least one of waiting requests of that group exceeds the maximum waiting time threshold and the same applied to all its copies.

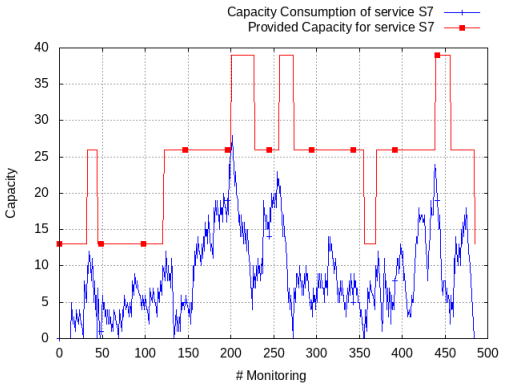
(e) A simulation script that specifies that the three provided strategies should be evaluated and compared on the SBP model using the elasticity controller generated from our elasticity model. Figure 6.7 illustrates the defined strategies and the simulation script.

6.3.2.3 Evaluation results

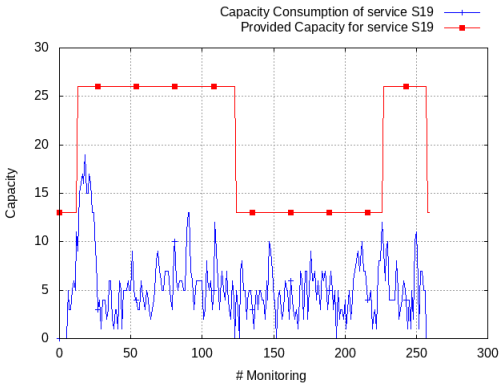
In order to evaluate elasticity strategies, we have defined some evaluation indicators that can be obtained from monitoring a given SBP model. In Figure 6.8 and Figure 6.9, we compare the amount of used capacity to the total provided capacity over time for a given service engine. The used capacity is computed by summing up the consumed capacity in each copy of the service engine. The provided capacity is computed by summing up the provided capacity for each service engine copy per strategy. The histogram in Figure 6.10 illustrates the rate of QoS violation for each strategy in the 2^{nd} evaluation scenario. We compute at each monitoring cycle the number of requests violated the QoS requirement. Then, we compute the violation rate by dividing the number of violations by the total number of process requests. The resulted plots for the first evaluation scenario show how the same strategy logic can behave differently for different process models even when it is used for the same workload. We can see the perfect adjustment of capacity for services in the SBP No. 3 and the SBP No. 4 while for other processes the plots shows more unused capacity which may indicate the necessity to adapt the strategy logic to the specificity of the process by, for example, changing thresholds or adding some conditions. On the other hand, the resulted plots for the second evaluation scenario show how each strategy is performed for both elasticity models which allows SBP holders to not only compare strategies but also elasticity models providing them with insight on which one of elasticity models (or actions) is more suitable for their process and their QoS requirements. The SBP holders can observe the elasticity behavior of their processes by analyzing the difference between the allocated and the consumed capacity and the violation rate of each strategy for different elasticity model. This analyses allows to make decision on adjusting the defined thresholds or changing some conditions in the elasticity strategy.

6.4 Conclusion

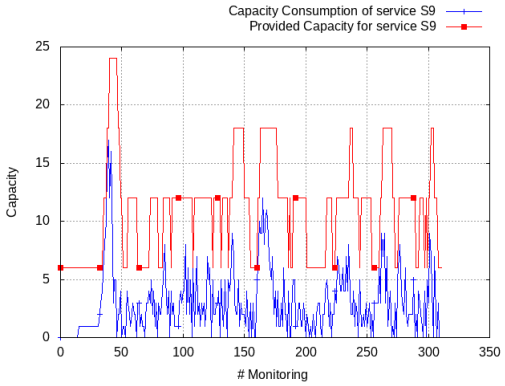
In this chapter, we presented a framework for describing and evaluating elasticity strategies for elastic SBPs, called StratFram. StratFram enables users through a set of editors for different DSLs to define their elastic systems by providing their elasticity



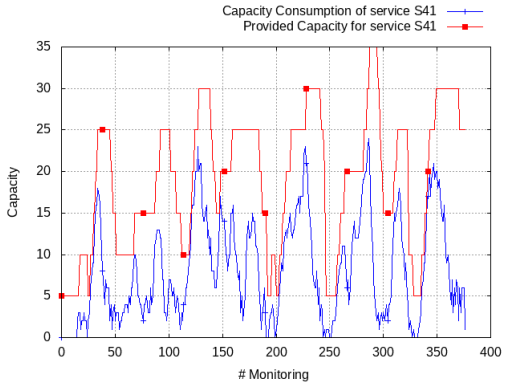
(a) SBP No. 1



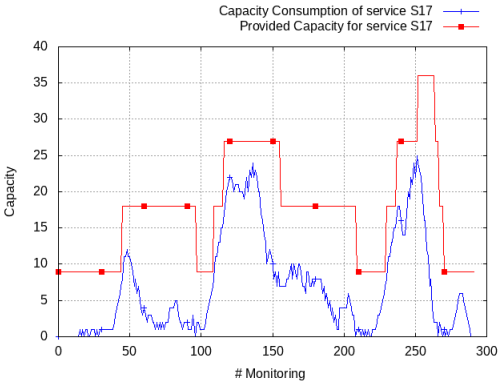
(b) SBP No. 2



(c) SBP No. 3

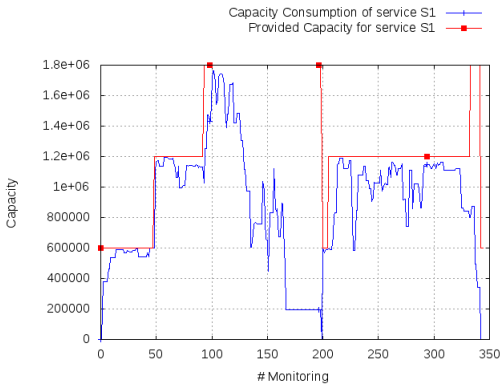


(d) SBP No.4

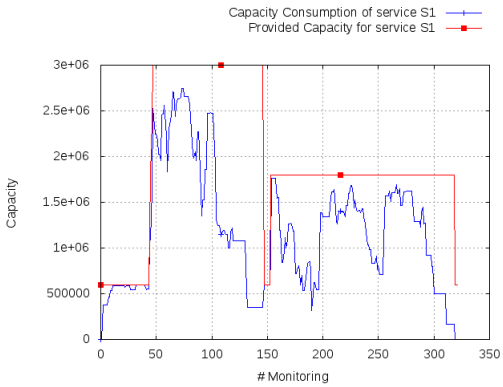


(e) SBP No. 5

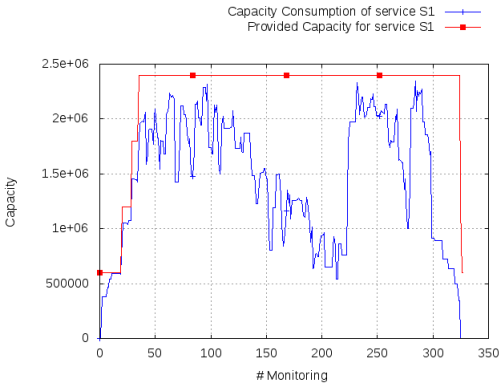
Figure 6.8: The evolution of capacity of a service in each SBP Models in the 1^{st} Evaluation Scenario



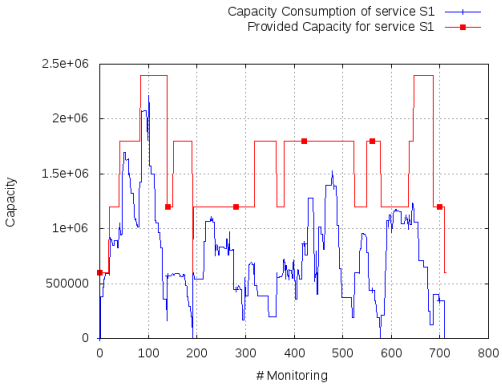
(a) Service S1 using StrategyResponseTime with SPEEDL elasticity model



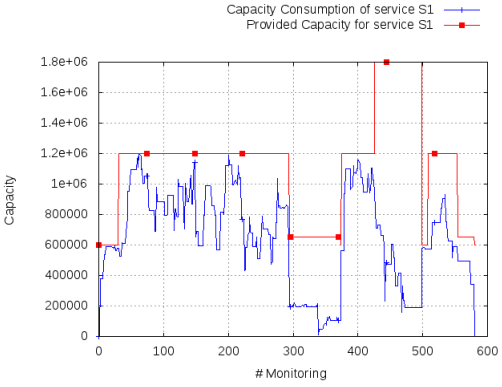
(b) Service S1 using StrategyWithMigration with SPEEDL elasticity model



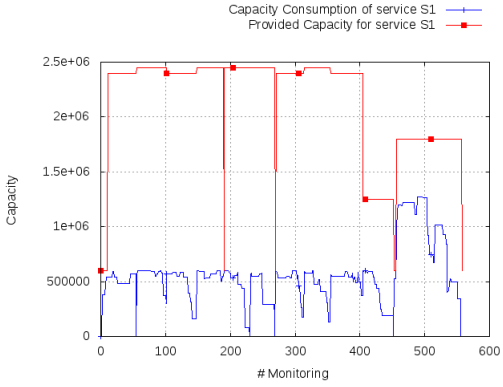
(c) Service S1 using StrategyWithMigrationGP with SPEEDL elasticity model



(d) Service S1 using RTWithCategories with Jrad *et al.* elasticity model

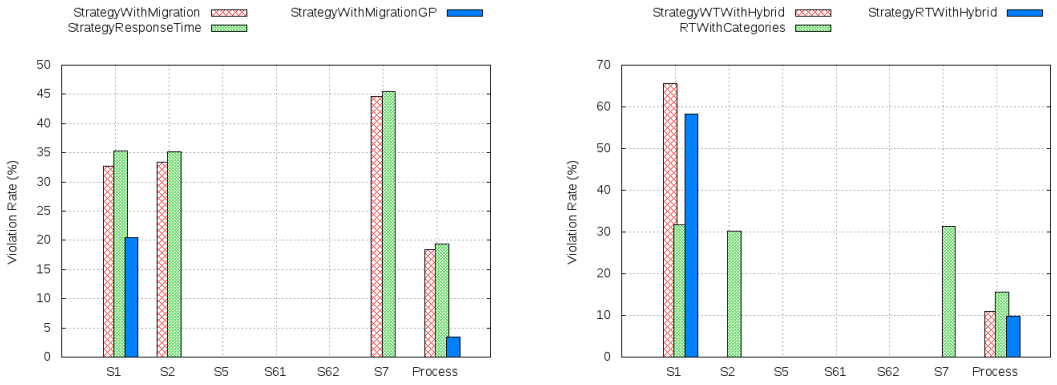


(e) Service S1 using StrategyRTWithHybrid with Jrad *et al.* elasticity model



(f) Service S1 using StrategyWTWithHybrid with Jrad *et al.* elasticity model

Figure 6.9: The evolution of capacity of SBP Model in the 2^{nd} Evaluation Scenario



(a) (b)

Figure 6.10: (a) Violation rate (%) in each elastic service engine and process using different strategies of evaluation project 1 (b) Violation rate (%) in each elastic service engine and process using different strategies of evaluation project 2

model along with their SBP models and strategies. It provides them with a set of languages that facilitate the description and the evaluation of elasticity strategies that can be based on different elasticity models. It has been designed in a way to separate the description part from the functional part of the framework. The description part which is represented by StratFram languages is designed to conceal the complexity and the type of underlying techniques and systems used for the evaluation which makes the framework extensible to other techniques and systems without affecting the users interaction with the framework and the previously defined scripts. We illustrated the use of our framework through two evaluation scenarios where we used in the first one publicly available SBPs while, in the second one, we defined two use cases for two different elasticity models.

CHAPTER 7

Conclusions and Future work

"This is not the end.
It is not even the beginning of the end.
But it is, perhaps, the end of the beginning."
Winston Churchill

This last chapter summarizes the main results of our current research, emphasizing the PhD thesis contributions. Furthermore, possible future research directions related to the research work presented in this thesis are outlined and discussed based on the challenges that were not fully addressed due to the limited time frame of the PhD thesis.

7.1 Conclusions

The thesis is based on the observation that there is a lack of solutions (both commercial and research) that allow SBP holders to describe non-trivial elasticity strategies for elastic SBPs, while considering their specific characteristics and different requirements, and to guarantee their effectiveness before putting them in use in the cloud. So, our main objective in this thesis was to provide SBP holders with a framework for defining and evaluating elasticity strategies that will be used to ensure the elasticity of their SBPs in the cloud.

As a first contribution toward this objective, we proposed a formal model for describing elastic execution environments of SBPs, on which the decisions of strategies will be applied. We argued that it is beneficial to employ formal methods for modeling elastic SBPs that allow analysing their elasticity behavior resulted from using a specific elasticity strategy. The proposed model defined based on timed-colored petri nets. It describes the characteristics of services engines, hosting SBP's services, as well as the characteristics of their requests. Using this model, SBPs holders are allowed to

define strategies using different elasticity indicators and to consider different requirements for requests. Besides the formal model, we defined and formalized two elasticity mechanisms (*i.e.*, duplication and consolidation) for hybrid scaling to show their application on our model for ensuring the elasticity of the execution environment of SBPs at service-level by allowing adding/removing service engine copies with different configurations. Applying the defined mechanisms/capabilities by an elasticity controller on our model according to an elasticity strategy changes its structure and status allowing the evaluation of the elasticity strategy before using it in a real Cloud environment.

Since the elasticity model is the basis for specifying elasticity strategies and constructing an elasticity controller that manages and evaluates the elasticity of SBPs, we proposed in this work StratModel language to allow describing elasticity models with different elasticity capabilities and to generate their corresponding elasticity controllers that can be used to evaluate strategies on our formal model. StratModel enables SBP holders to define their elasticity models by specifying customized metrics, properties and elasticity actions. An action mechanism is provided through a set of examples illustrating how the generated controller should operate when applies the action that changes the structure of the managed SBP. The customized controller is generated by StratModel using a pre-defined template described using high-level petri net to allow the evaluation of strategies.

Based on StratModel language, we designed Strat, a rule-based domain specific language, for describing elasticity strategies governing SBP elasticity. It enables SBP holders to specify QoS requirements of a SBP at different granularity levels (*i.e.*, process, service, and instance level) with taking into account the fundamental characteristics of elastic SBPs. It relies on StratModel to define the elasticity model on which its grammar will be adapted. So, when using Strat to define strategies, SBP holders will be provided with only the elasticity capabilities that are specified in the StratModel document.

After developing our languages, a framework supporting SBPs holders in choosing a proper elasticity strategy to be used during the configuration phase of their elastic SBPs was developed. StratFram, the framework for describing and evaluating elasticity strategies for elastic SBPs, considers evaluating, through simulation, different elasticity strategies for different elasticity models based on our formal model for elastic SBPs. It has been designed based on a set of languages including StratModel and Strat for describing respectively elasticity models and elasticity strategies along with two other languages where the first one, named SBP, is for describing elastic execution environments of SBPs and generating their corresponding formal models and the second one, named StratSim, is for specifying simulation properties/elements used for the evaluation. These languages which represent the descriptive part of the framework were designed to facilitate the description of the elastic system to be evaluated while concealing the implementation complexity and the type of underlying techniques/systems used for the evaluation. The evaluation of elasticity strategies using StratFram consists in providing a set of plots that allows the analysis and the comparison of strate-

gies to choose the effective ones. The use of StratFram has been illustrated through two evaluation scenarios where the first one was based on publicly available SBPs while, the second one described two use cases for two different elasticity models.

7.2 Future work

In this thesis, we faced different complex problems in order to provide a framework for evaluating elasticity strategies for elastic SBPs. As a future work, we envision extending the framework, or the used model, for addressing remaining issues. The following research perspectives are feasible for future work in short, medium and long term.

- As short term future work, we plan to extend our formal model by incorporating other properties/aspects related to data storage and network. In our modeling approach, we mainly focused on describing the characteristics of service engines, hosting services in SBPs, provisioned from one single cloud provider as well as the characteristics of services requests. However, some other properties, such as network latency, time and cost of data exchange between service engines and external database, should be also considered to allow the model to cover more possible cases occurred in the cloud.

- As medium term work, we aim to extend StratFram by providing other implementations for different systems. Our goal behind this extension is to allow SBP holders to evaluate their strategies not only using formal method such as petri nets but also to simulate and analyze their performance using a business process engine such as Activiti[1] or evaluating them in real cloud environments as well using Open Cloud Computing Interface (OCCI) [Nyren 2011]. So, they will be able to choose between different ways to evaluate their strategies.

- Finally, as long term future work, we will work on developing a domain-specific model checker that considers the specificity of elastic SBPs. This model checker will be integrated in our framework to allow the formal verification of properties. StratFram was developed based on formal methods to formally evaluate and verify the correctness of strategies using especially model-checkers. However, the existing model-checkers are incapable of manipulating models that are changeable in structure due to the applications of elasticity mechanisms.

[1] https://www.activiti.org/

Bibliography

[Aktas 2015] A. Aktas, S. Cebi and I. Temiz. *A new evaluation model for service quality of health care systems based on AHP and information axiom.* Journal of Intelligent and Fuzzy Systems, vol. 28, no. 3, pages 1009–1021, 2015. (Cited on page 27.)

[Ali-Eldin 2012] A. Ali-Eldin, J. Tordsson and E. Elmroth. *An adaptive hybrid elasticity controller for cloud infrastructures.* In NOMS, pages 204–21, 2012. (Cited on pages xi, 29, 30 and 43.)

[Allweyer 2010] Th. Allweyer. Bpmn 2.0. BoD, 2010. (Cited on pages 22 and 24.)

[Ameller 227] D. Ameller and X. Franch. *Service Level Agreement Monitor (SALMon).* In Proceedings of the Seventh International Conference on Composition-Based Software Systems, ICCBSS'2008, page 2008, Madrid, Spain, 224-227. (Cited on page 19.)

[Amziani 2013] M. Amziani, T. Melliti and S. Tata. *Formal Modeling and Evaluation of Stateful Service-Based Business Process Elasticity in the Cloud.* In CoopIS, pages 21–38, 2013. (Cited on pages 102 and 103.)

[Amziani 2015] M. Amziani. *Modeling, Evaluation and Provisioning of Elastic Service-based Business Processes in the Cloud.* PhD thesis, Telecom Sudparis et l'University Evry Val d'Essonne, Paris, France, 2015. (Cited on pages xi, 3, 39, 40, 43 and 44.)

[Backus 1959] J. W. Backus. *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference.* In IFIP Congress, pages 125–131, 1959. (Cited on page 68.)

[Bessai 2012a] K. Bessai, S. Youcef, A. Oulamara, C. Godart and S. Nurcan. *Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments.* In Proceedings of the 5th International Conference on Cloud Computing, CLOUD'2012, Honolulu, Hawaii, USA, 2012. (Cited on pages 3, 28 and 43.)

[Bessai 2012b] K. Bessai, S. Youcef, A. Oulamara, C. Godart and S. Nurcan. *Resources allocation and scheduling approaches for business process applications in Cloud contexts*. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science, CloudCom'2012, pages 496–503, Taipei, Taiwan, 2012. (Cited on pages 3, 28 and 43.)

[Bikas 2016] M.A.N. Bikas, A. Alourani and M. Grechanik. *How Elasticity Property Plays an Important Role in the Cloud: A Survey*. In Advances in Computers, volume 103, pages 1–30. 2016. (Cited on page 17.)

[Cearley 2010] D. W. Cearley. *Cloud Computing*. Technical report, Gartner, 2010. (Cited on page 14.)

[Copil 2013] G. Copil, D. Moldovan, T. Hong-Linh and S. Dustdar. *SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications*. In CCGRID, pages 112–119, 2013. (Cited on pages xiii, 34, 35, 43 and 44.)

[Copil 2015] G. Copil, H. L. Truong, D. Moldovan, S. Dustdar, D. Trihinas, G. Pallis and M. D. Dikaiakos. *Evaluating Cloud Service Elasticity Behavior*. International Journal of Cooperative Information Systems, vol. 24, no. 3, 2015. (Cited on pages xi, 37, 38, 43 and 44.)

[Copil 2016] G. Copil, D. Moldovan, H. L. Truong and S. Dustdar. *Continuous elasticity: Design and operation of elastic systems*. it - Information Technology, vol. 58, no. 6, pages 329–348, 2016. (Cited on page 4.)

[Cur 1998] Sap r/3 business blueprint: Understanding the business process reference model. Inc. Prentice-Hall, 1998. (Cited on page 102.)

[Cypher 1993] A. Cypher, editor. Watch what i do – programming by demonstration. MIT Press, Cambridge, MA, USA, 1993. (Cited on page 73.)

[Czarnecki 2000] K. Czarnecki and U.W. Eisenecker. Generative programming: Methods, techniques, and application, volume 1st edition. Addison-Wesley, 2000. (Cited on page 68.)

[Davenport 1993] Th. H. Davenport. Process innovation - reengineering work through information technology. Harvard Business School Press, 1993. (Cited on page 22.)

[Deursen 2000] A. V. Deursen, P. Klint and J. Visser. *Domain-specific Languages: An Annotated Bibliography*. ACM SIGPLAN Notices, vol. 35, no. 6, pages 26–36, 2000. (Cited on page 68.)

[Dustdar 2011] S. Dustdar, Y. Guo, B. Satzger and H. L. Truong. *Principles of Elastic Processes*. IEEE Internet Computing, vol. 15, no. 5, pages 66–71, 2011. (Cited on pages 3, 5 and 25.)

[Dutreilh 2010] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre and I. Truck. *From Data Center Resource Allocation to Control Theory and Back*. In Proceedings of the 3rd IEEE International Conference on Cloud Computing, CLOUD'2010, pages 410–417, Miami, Florida, US, 2010. (Cited on page 4.)

[Earl 1994] M. J. Earl. *The new and the old of business process redesign*. The Journal of Strategic Information Systems, vol. 3, no. 1, pages 5–22, 1994. (Cited on page 22.)

[Emeakaroha 2010] V. C. Emeakaroha, I. Brandic, M. Maurer and S. Dustdar. *Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments*. In Proceedings of the 2010 International Conference on High Performance Computing and Simulation, HPCS'2010, pages 48–54, Caen, France, 2010. (Cited on page 19.)

[Evangelidis 2018] A. Evangelidis, D. Parker and R. Bahsoon. *Performance modelling and verification of cloud-based auto-scaling policies*. Future Generation Computer Systems, vol. 87, pages 629–638, 2018. (Cited on pages 41 and 43.)

[Farokhi 2016] S. Farokhi, P. Jamshidi, E. B. Lakew, I. Brandic and E. Elmroth. *A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach*. Future Generation Computer Systems, vol. 65, pages 57–72, 2016. (Cited on pages xi, 31, 32 and 43.)

[Frey 2013] S. Frey, C. Luthje and Ch. Reich. *Key Performance Indicators for Cloud Computing SLAs*. In Proceedings of The Fifth International Conference on Emerging Network Intelligence, EMERGING'13, pages 60–64, Porto, Portugal, 2013. (Cited on page 19.)

[Galante 2012] G. Galante and L. C. E. d. Bona. *A Survey on Cloud Computing Elasticity*. In Proceedings of the fifth IEEE International Conference on Utility and Cloud Computing, UCC'2012, pages 263–270, 2012. (Cited on page 18.)

[Garcia-Recuero 2014] A. Garcia-Recuero, S. Esteves and L. Veiga. *Towards quality-of-service driven consistency for Big Data management*. International Journal of Big Data Intelligence, vol. 1, no. 1-2, pages 74–88, 2014. (Cited on page 27.)

[Garg 2013] S. K. Garg, S. Versteeg and R. Buyya. *A framework for ranking of cloud computing services*. Future Generation Computer Systems, vol. 29, no. 4, pages 1012–1023, 2013. (Cited on page 17.)

[Gens 2008] F. Gens. *Defining Cloud Services and Cloud Computing*. Technical report, IDC exchange, 2008. (Cited on page 14.)

[Hammer 1993] M. Hammer and J. Champy. Reengineering the corporation: A manifesto for business revolution. Nicholas Brealey Publishing, London, UK, 1993. (Cited on page 21.)

[Han 2014] 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, pages 82 – 98, 2014. (Cited on page 17.)

[Heidari 2014] F. Heidari and P. Loucopoulos. *Quality Evaluation Framework (QEF): Modeling and Evaluating Quality of Business Processes.* International Journal of Accounting Information Systems, vol. 15, no. 3, pages 193–223, 2014. (Cited on page 19.)

[Herbst 2013] N. R. Herbst, S. Kounev and R. Reussner. *Elasticity in Cloud Computing: What It Is, and What It Is Not.* In ICAC, pages 23–27, 2013. (Cited on pages 2 and 17.)

[Hoenisch 2014] Ph. Hoenisch. *ViePEP - A BPMS for Elastic Processes.* In Proceedings of the 6th Central European Workshop on Services and their Composition, ZEUS'2014, Potsdam, Germany, 2014. (Cited on pages xi, 3, 28, 29, 43 and 44.)

[Hoffa 2008] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman and J. Good. *On the Use of Cloud Computing for Scientific Workflows.* In Proceedings of the 4th IEEE International Conference on eScience, eScience'2008, pages 640–645, Indianapolis, USA, 2008. (Cited on pages 5 and 24.)

[Hogan 2011] M. Hogan, F. Liu, A. Sokole and J. Tong. *NIST Cloud Computing Standards Roadmap.* Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011. (Cited on pages xi, 1, 14, 15 and 16.)

[IBM 2005] *An Architectural Blueprint for Autonomic Computing.* Technical report, IBM, 2005. (Cited on page 31.)

[IDG 2018] IDG. *2018 Cloud Computing Survey.* https://www.idg.com/tools-for-marketers/2018-cloud-computing-survey/, 2018. (Cited on page 2.)

[Jacob 2004] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir and A. F. Yassin. A practical guide to the ibm autonomic computing toolkit. IBM redbooks. IBM Corporation, International Technical Support Organization, 2004. (Cited on pages 28 and 78.)

[Jacobson 1994] I. Jacobson, M. Ericsson and A. Jacobson. The object advantage: Business process reengineering with object technology. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1994. (Cited on page 22.)

[Jansen 2001] T. Jansen and I. Wegener. *Evolutionary Algorithms - How to Cope with Plateaus of Constant Fitness and when to Reject Strings of the Same Fitness.* TEVC, vol. 5, no. 6, pages 589–599, 2001. (Cited on page 6.)

[Jensen 1991] K. Jensen and G. Rozenberg. High-level petri nets: Theory and application. Springer-Verlag, London, UK, 1991. (Cited on pages 52 and 78.)

[Jensen 2009] K. Jensen and L. M. Kristensen. Coloured petri nets: Modelling and validation of concurrent systems. Springer Publishing Company, Incorporated, 1st édition, 2009. (Cited on pages 50 and 55.)

[Jrad 2016] A. B. Jrad, S. Bhiri and S. Tata. *Description and Evaluation of Elasticity Strategies for Business processes in the Cloud*. In Proceedings of the IEEE International Conference on Services Computing (SCC), pages 203–210, San francisco, California, USA, 2016. (Cited on page 10.)

[Jrad 2017a] A. B. Jrad, S. Bhiri and S. Tata. *Data-Aware Modeling Of Elastic Processes For Elasticity Strategies Evaluation*. In Proceedings of the IEEE 10th International Conference on Cloud Computing (CLOUD), pages 464–471, Honolulu, Hawaii, USA, 2017. (Cited on page 9.)

[Jrad 2017b] A. B. Jrad, S. Bhiri and S. Tata. *STRATModel: Elasticity Model Description Language for Evaluating Elasticity Strategies for Business Processes*. In On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE, pages 448–466, Rhodes, Greece, 2017. (Cited on pages 9 and 10.)

[Jrad 2019] A. B. Jrad, S. Bhiri and S. Tata. *STRATFram: A framework for describing and evaluating elasticity strategies for service-based business processes in the cloud*. Future Generation Computer Systems, vol. 97, pages 69–89, 2019. (Cited on page 11.)

[Juric 2006] M. B. Juric. Business process execution language for web services bpel and bpel4ws 2nd edition. Packt Publishing, 2006. (Cited on pages 22 and 24.)

[Juve 2010] G. Juve and E. Deelman. *Scientific Workflows and Clouds*. ACM Crossroads, vol. 16, no. 3, pages 14–18, 2010. (Cited on pages 5 and 24.)

[K. 2015] K., Q. Wang, J. Hur, K.-J. Park and L. Sha. *Medical-Grade Quality of Service for Real-Time Mobile Healthcare*. Computer, vol. 48, no. 2, pages 41–49, 2015. (Cited on page 27.)

[Katoh 2008] K. Katoh and H. Toh. *Recent developments in the MAFFT multiple sequence alignment program*. Briefings in Bioinformatics, vol. 9, no. 4, pages 286–298, 2008. (Cited on page 6.)

[Kenagy 1999] J. W. Kenagy, D. M. Berwick and M. F. Shore. *Service Quality in Health Care*. The Journal of the American Medical Association (JAMA), vol. 281, no. 7, pages 661–665, 1999. (Cited on page 27.)

[Klems 2009] M. Klems, J. Nimis and S. Tai. *Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing*. In Proceedings of the 7th Workshop on E-Business: Designing E-Business Systems. Markets, Services, and Networks, pages 110–123, Paris, France, 2009. (Cited on page 14.)

[Kritikos 2014] K. Kritikos, J. Domaschka and A. Rossini. *SRL: A Scalability Rule Language for Multi-cloud Environments*. In Proceedings of the IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom'2014, pages 1–9, Singapore, 2014. (Cited on pages xi, 34, 36, 43 and 44.)

[Li 2011] M. Li, F. Ye, M. Kim, H. Chen and H. Lei. *A Scalable and Elastic Publish/-Subscribe Service*. In Proceedings of the 25th IEEE International Parallel Distributed Processing Symposium, IPDPS, pages 1254–1265, Alaska, USA, 2011. (Cited on page 17.)

[Lieberman 2001] H. Lieberman, editor. Your wish is my command: Programming by example. organ Kaufmann Publishers Inc., 2001. (Cited on page 73.)

[Liu 2000] L. Liu, C. PU, K. Schwan and J. Walpole. *InfoFilter: Supporting quality of service for fresh information delivery*. New Generation Computing, 2000. (Cited on page 27.)

[Liu 2016] Y. Liu, D. Gureya, A. Al-Shishtawy and V. Vlassov. *OnlineElastMan: Self-Trained Proactive Elasticity Manager for Cloud-Based Storage Services*. In ICCAC, 2016. (Cited on pages xi, 30, 32 and 43.)

[Loff 2014] J. Loff and J. Garcia. *Vadara: Predictive Elasticity for Cloud Applications*. In Proceedings of the IEEE 6th International Conference on Cloud Computing Technology and Science, pages 541–546, 2014. (Cited on pages xi, 29, 31 and 43.)

[Mans 2010] R. S. Mans, N. C. Russell, W. M. P. van der Aalst, A. J. Moleman and P. J. M. Bakker. Transactions on petri nets and other models of concurrency iv, chapter Schedule-Aware Workflow Management Systems, pages 121–143. Springer Berlin Heidelberg, 2010. (Cited on pages 2 and 25.)

[Mell 2011] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Technical report, National Institute of Standards and Technology, 2011. (Cited on page 17.)

[Mendling 2008] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst and G. Neumann. *Detection and prediction of errors in EPCs of the SAP reference model*. Data & Knowledge Engineering, vol. 64, no. 1, pages 312–329, 2008. (Cited on page 102.)

[Merkel 2014] D. Merkel. *Docker: Lightweight Linux Containers for Consistent Development and Deployment*. Linux Journal, vol. 2014, no. 239, 2014. (Cited on pages 47 and 93.)

[Mernik 2005] M. Mernik, J. Heering and A. M. Sloane. *When and How to Develop Domain-specific Languages*. ACM Computing Surveys, vol. 37, no. 4, pages 316–344, 2005. (Cited on page 68.)

[Moltó 2016] G. Moltó, M. Caballer and C. de Alfonso. *Automatic memory-based vertical elasticity and oversubscription on cloud platforms*. Future Generation Computer Systems, vol. 56, pages 1–10, 2016. (Cited on pages xi, 32, 33 and 43.)

[Murata 1989] T. Murata. *Petri nets: Properties, analysis and applications*. In Proceedings of the IEEE, volume 77, pages 541–580, 1989. (Cited on pages 24 and 49.)

[Naskos 2015a] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou and S. Sioutas. *Dependable Horizontal Scaling Based on Probabilistic Model Checking*. In Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid'2015, pages 31–40, Shenzhen, China, 2015. (Cited on page 4.)

[Naskos 2015b] A. Naskos, E. Stachtiari, P. Katsaros and A. Gounaris. *Probabilistic Model Checking at Runtime for the Provisioning of Cloud Resources*. In Proceedings of the 6th International Conference on Runtime Verification, pages 275–280, 2015. (Cited on pages xi, 38, 39 and 43.)

[Nyren 2011] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch. *Open Cloud Computing Interface - Core*. Technical report, Open Grid Forum (OGF), 2011. (Cited on page 115.)

[Ocaña 2012] K. A. C. S. Ocaña, D. d. Oliveira, F. Horta, J. Dias, E. S. Ogasawara and M. Mattoso. *Exploring Molecular Evolution Reconstruction Using a Parallel Cloud Based Scientific Workflow*. In BSB, 2012. (Cited on page 6.)

[Ould 1995] M. A. Ould. Business processes: Modelling and analysis for re-engineering and improvement. West Sussex, England: John Wiley and Sons, 1995. (Cited on page 22.)

[Perez-Sorrosal 2011] F. Perez-Sorrosal, M. Pati no Martinez, R. Jimenez-Peris and B. Kemme. *Elastic SI-Cache: Consistent and Scalable Caching in Multi-tier Architectures*. The VLDB Journal, vol. 20, no. 6, pages 841–865, 2011. (Cited on page 17.)

[Pommereau 2015] F. Pommereau. *SNAKES: A Flexible High-Level Petri Nets Library (Tool Paper)*. In Proceedings of the 36th International Conference on Application and Theory of Petri Nets and Concurrency, pages 254–265, Brussels, Belgium, 2015. (Cited on pages 100 and 102.)

[Ray 1999] P. Ray and G. Weerakkody. *Quality of service management in health care organizations: a case study*. In Proceedings of the 12th IEEE Symposium on Computer-Based Medical Systems, CBMS'99, pages 80–85, Stamford, CT, USA, 1999. (Cited on page 27.)

[Rohjans 2012] S. Rohjans, C. Dänekas and M. Uslar. *Requirements for Smart Grid ICT-architectures*. In Proceedings of the 2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe), pages 1–8, 2012. (Cited on pages 2 and 25.)

[Rosinosky 2016] G. Rosinosky, S. Youcef and F. Charoy. *An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing environment*. In Proceedings of the 9th IEEE International Conference on Cloud Computing, CLOUD'2016, San Francisco, USA, 2016. (Cited on page 3.)

[Ross 1978] D. T. Ross. *Origins of the APT Language for Automatically Programmed Tools*. ACM SIGPLAN Notices - Special issue: History of programming languages conference, vol. 13, no. 8, pages 61–99, 1978. (Cited on page 68.)

[Ruparelia 2016] N. B. Ruparelia. Cloud computing. The MIT Press Essential Knowledge series, 2016. (Cited on page 14.)

[Sandhu 2015] R. Sandhu and S. K. Sood. *Scheduling of big data applications on distributed cloud based on QoS parameters*. Cluster Computing, vol. 18, no. 2, pages 817–828, 2015. (Cited on page 27.)

[Schouten 2012] E. Schouten. *Rapid Elasticity and the Cloud*. https://www.ibm.com/blogs/cloud-computing/2012/09/12/rapid-elasticity-and-the-cloud/, September 2012. (Cited on page 17.)

[Schulte 2014] S. Schulte, Ch. Janiesch, S. Venugopal, I. Weber and Ph. Hoenisch. *Elastic Business Process Management: State of the Art and Open Challenges for BPM in the Cloud*. Future Generation Computer Systems, 2014. (Cited on pages 2 and 25.)

[Shawky 2012] D. M. Shawky and A. F. Ali. *Defining a measure of cloud computing booktitle=Proceedings of the 2012 1st International Conference on Systems and Computer Science (ICSCS), elasticity*. pages 1–5, 2012. (Cited on page 17.)

[Smith 2005] T. Smith. *Quality of service requirements for system wide information management (SWIM)*. In Proceedings of the 24th Digital Avionics Systems Conference, volume 1 of *DASC'2005*, pages 1–8, Hyatt Regency Crystal City, Washington, D.C., 2005. (Cited on page 27.)

[Stamatakis 2014] A. Stamatakis. *RAxML Version 8: A Tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies*. Bioinformatic, vol. 30, no. 9, 2014. (Cited on page 6.)

[Suleiman 2013] B. Suleiman and S. Venugopal. *Modeling Performance of Elasticity Rules for Cloud-Based Applications*. In EDOC, pages 201–206, 2013. (Cited on pages xi, 38, 40 and 43.)

[van der Aalst 1998] W. M. P. van der Aalst. *The application of perti nets to workflow management*. Journal of Circuits, Systems, and Computers, vol. 8, no. 1, pages 21–66, 1998. (Cited on page 49.)

[van der Aalst 2011] W.M.P. van der Aalst and Ch. Stahl. Modeling business processes: A petri net-oriented approach. The MIT Press, 2011. (Cited on page 49.)

[Vaquero 2009] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner. *A Break in the Clouds: Towards a Cloud Definition*. ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pages 50–55, 2009. (Cited on page 14.)

[Wang 2008] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer and W. Karl. *Scientific Cloud Computing: Early Definition and Experience*. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, HPCC'08, pages 825–830, Dalian, China, 2008. (Cited on pages 14 and 16.)

[Weske 2012] M. Weske. Business process management - concepts, languages, architectures, volume 2nd edition. Springer, 2012. (Cited on pages 3 and 22.)

[Wile 2004] D. Wile. *Lessons Learned from real DSL Experiments*. Science of Computer Programming, vol. 51, no. 3, pages 265–290, 2004. (Cited on page 68.)

[Xu 2012] W. Xu, Z. Zhou, D.T. Pham, Q. Liu, C. Ji and W. Meng. *Quality of service in manufacturing networks: a service framework and its implementation*. The International Journal of Advanced Manufacturing Technology, vol. 63, no. 9-12, pages 1227–1237, 2012. (Cited on page 27.)

[Yangui 2011] S. Yangui, M. Mohamed, S. Tata and S. Moalla. *Scalable service containers*. In CloudCom, pages 348–356, 2011. (Cited on pages 47 and 93.)

[Zabolotnyi 2015] R. Zabolotnyi, Ph. Leitner, S. Schulte and S. Dustdar. *SPEEDL - A Declarative Event-Based Language for Cloud Scaling Definition*. In IEEEServices, 2015. (Cited on pages xiii, 35, 37, 43, 44 and 105.)

[Zhou 2009] Z. Zhou, W. Xu, D.T. Pham and C. Ji. *QoS modeling and analysis for manufacturing networks: A service framework*. In Proceedings of the 7th IEEE International Conference on Industrial Informatics, INDIN'2009, pages 825–830, Cardiff, Wales, UK, 2009. (Cited on page 27.)

[Zloof 1975] M. M. Zloof. *Query By Example*. In Proceedings of National Compute Conference, pages 431–438. AFIPS Press, 1975. (Cited on page 73.)