FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Cloud Services Elasticity Control: from requirements specification to operations management

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Dipl. Ing. Georgiana Copil
Matrikelnummer 1227555

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar
Zweitbetreuung: Priv.-Doz. Dr. Hong-Linh Truong

Diese Dissertation haben begutachtet:

Univ. Prof. Schahram Dustdar	Univ. Prof. Nectarios Koziris

Wien, 12. Februar 2016

Georgiana Copil

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Cloud Services Elasticity Control: from requirements specification to operations management

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Dipl. Ing. Georgiana Copil
Registration Number 1227555

to the Faculty of Informatics
at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar
Second advisor: Priv.-Doz. Dr. Hong-Linh Truong

The dissertation has been reviewed by:

Univ. Prof. Schahram Dustdar	Univ. Prof. Nectarios Koziris

Vienna, 12th February, 2016

Georgiana Copil

Erklärung zur Verfassung der Arbeit

Dipl. Ing. Georgiana Copil
Wagramerstrasse 56, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. Februar 2016

Georgiana Copil

Acknowledgements

This work was supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790) and by TU Wien's Adaptive Distributed Systems Doctoral College.

I am very grateful to my adviser, Professor Schahram Dustdar and my co-adviser, Priv.-Doz. Hong-Linh Truong for having me in Distributed Systems Group (DSG), and for their continuous support to achieve this work. Many thanks to the DSG colleagues, for the very nice work environment, and for the many motivational chats that every PhD student needs. I owe a great deal of what I am today to the group where I had my first contacts with research, DSRL of Cluj-Napoca, led by Professor Ioan Salomie.

I would like to thank the CELAR team, for the many telcos, debates and feedback, especially to the LinC team of University of Cyprus for a very nice collaboration to achieve part of this thesis work.

I am very thankful to Professor Nectarios Koziris, for being my examiner.

Finally, I thank my family, for their unconditional support. This thesis is dedicated to you.

Kurzfassung

Utility basiertes Computing, wie etwa Cloud Computing, ist eine treibende Kraft für die Etablierung stark verteilter Anwendungen (z.B., microservice-basierte Anwendungen). Cloud Computing ermöglicht es unterschiedlichen Gruppen von Akteuren aus einer Vielzahl von Konfigurationen zu wählen um ihre Anwendungen zu betreiben (z.B., öffentlich-private bzw. multi-Cloud Bereitstellung). Zusätzlich können Applikationen an unterschiedliche Technologie- und Geschäftsperspektiven optimal angepasst werden. Nach Entwicklung der Anwendung können Interessenvertreter entscheiden ihre Anwendung als Cloud Dienste (d.h., Software als Service) anzubieten und profitieren damit von den bereitgestellten Cloud Diensten (die z.B., Ressourcen je nach Bedarf bereitzustellen in der Form von Infrastruktur als Service oder Plattform als Service). Um die ganze Bandbreite an Cloud Diensten zu nutzen, müssen Interessengruppen ihre Anwendungen entsprechend anpassen und dabei die Last, sowie die Anforderungen der Benutzer, berücksichtigen. Dabei ist es wichtig die gewünschte Qualität zu einem möglichst niedrigen Preis für die bereitgestellten Cloud Ressourcen zu erreichen. Um jedoch diese Vorgaben zu erfüllen, müssen Betreiber solcher Anwendungen entweder fachkundige Personen, die Überwachung und Anpassung übernehmen, bezahlen, oder vorhandene Lösungen die von Cloud Anbietern bereitgestellt werden, verwenden. Diese Lösungen bieten allerdings nur einfache Skalierungsmöglichkeiten, die es nicht erlauben komplexe Anwendungen, Anwendungseinstellungen, und Abhängigkeiten zwischen Anwendungskomponenten zu betrachten.

In dieser Arbeit wird ein Framework zur Elastizitätssteuerung von Cloud Diensten vorgestellt. Beginnend mit den Elastizitätsanforderungen wird die Sprache SYBL vorgestellt. SYBL unterstützt Akteure bei der Beschreibung von Anwendungsanforderungen durch Verwendung mehrerer Abstraktionsgrade. Mit dieser Sprache als Grundlage wird das Werkzeug rSYBL vorgestellt um die Elastizität von Cloud Diensten zu steuern. Dabei werden Verhalten sowie spezifizierte Anforderungen betrachtet. Um diese Steuerung zu verbessern, wird das Verhalten von Cloud Diensten über die Zeit und mittels mehrfacher Abstraktionsgrade geschätzt, welche auf Mechanismen basieren, die ebenfalls in rSYBL integriert werden. Das Framework wird weiter ausgebaut um das Operationsmanagement zur Laufzeit zu unterstützen und dabei Elastizitätsinteressen zu berücksichtigen. Dies erlaubt die Integration von unterschiedlichen Akteuren in den ganzen Elastizitätskontrollprozess. Unter Verwendung eines illustrativen Szenarios wird jeder der vorgeschlagenen Mechanismen überprüft und analysiert. Die gewonnenen Er-

gebnisse zeigen, dass rSYBL in der Lage ist, während der Laufzeit, Cloud Dienste, unter Verwendung von Überwachungsinformationen und Elastizitätsanforderungen, zu steuern.

Abstract

Utility based computing, such as cloud computing, is a driving force for the adoption of highly distributed applications (e.g., microservices-based applications). Through the use of cloud computing, application stakeholders can choose from a multitude of configurations for their application deployment, e.g., public-private deployments, multi-cloud deployments, and can adapt to whatever is best for their application from the technology and business perspectives. After the application deployment, stakeholders may choose to offer the application as a cloud service (i.e., Software as a Service), benefiting from cloud providers services (e.g., resources on-demand offered by Infrastructure as a Service providers or platforms on-demand offered by Platform as a Service providers). For fully using cloud-offered services, the application needs to be adapted by its stakeholders, considering the load, and their users' requirements, to obtain the desired quality at the minimum price to be paid to cloud providers in exchange for their resources. However, for achieving this, the application owner needs to pay a specialized person for monitoring and adapting whenever needed the application, or to use existing solutions offered by cloud providers which enable solely simple scaling, without considering complex applications, application-level configurations, and dependencies among application components.

In this thesis, a framework for elasticity control of cloud services is proposed. Starting from high-level requirements, a language, SYBL, is proposed, which supports service stakeholders in describing their requirements at multiple levels of abstraction. With this language as a basis, a framework, rSYBL, is proposed, in order to control cloud service elasticity considering its behavior and the specified requirements. For improving the control, the behavior of cloud services, in time, at multiple levels of abstraction, is estimated, based on mechanisms that are also integrated in rSYBL. The framework is further extended for supporting operations management at runtime, considering elasticity concerns, integrating stakeholders/employees in the whole control process. Using an illustrative case study, each of the proposed mechanisms is evaluated. Results show that rSYBL is able to control cloud services, during runtime using monitoring information and requirements coming from stakeholders.

Contents

List of Figures

List of Tables

Publications

This thesis is based on work published in scientific conferences, workshops, journals and books. For reasons of brevity, these core papers, which build the foundation of this thesis, are listed here once, and will generally not be explicitly referenced again. Parts of these papers are contained in verbatim.

1. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar. "rSYBL: a Framework for Specifying and Controlling Cloud Services Elasticity", ACM Transactions on Internet Technology (TOIT),2015.

2. Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, Marios Dikaiakos. "Evaluating Cloud Service Elasticity Behavior", International Journal of Cooperative Information Systems, 2015 (invited).

3. Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "Supporting Cloud Service Operation Management for Elasticity", the 13th International Conference on Service Oriented Computing. Goa, India, 16-19 November, 2015

4. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "On Controlling Cloud Services Elasticity in Heterogeneous Clouds", 6th Cloud Control Workshop, 7th IEEE/ACM International Conference on Utility and Cloud Computing, 8-11 December, London, 2014.

5. Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, Marios Dikaiakos. "ADVISE - a Framework for Evaluating Cloud Service Elasticity Behavior" the 12th International Conference on Service Oriented Computing. Paris, France, 3-6 November, 2014. **(Best paper award)**

6. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "Multi-Level Elasticity Control of Cloud Services", the 11th International Conference on Service Oriented Computing. Berlin, Germany, on 2-5 December, 2013.

7. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "SYBL+ MELA: Specifying, Monitoring, and Controlling Elasticity of Cloud Services", the 11th International Conference on Service Oriented Computing. Berlin, Germany, on 2-5 December, 2013.

8. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "SYBL: an Extensible Language for Controlling Elasticity in Cloud Applications", 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 14-16, 2013, Delft, the Netherlands.

9. Georgiana Copil, Daniel Moldovan, Duc-Hung Le, Hong-Linh Truong, Schahram Dustdar, Chrystalla Sofokleous, Nicholas Loulloudes, Demetris Trihinas, George Pallis, Marios D. Dikaiakos, Craig Sheridan, Evangelos Floros, Christos KK Loverdos, Kam Star, Wei Xing, On Controlling Elasticity of Cloud Applications in CELAR, Emerging Research in Cloud Distributed Computing Systems, Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series.

CHAPTER 1

Introduction

Cloud computing has been receiving significant attention over the last years. The computing world has been progressing towards the dream of having computing as utility [103], starting from shared computing models and virtualization as a time sharing mechanism, e.g., IBM's Remote Job Entry [16] and VM OS [87] from the 70s, to today's cloud computing model. According to NIST, *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.* [86]. The cloud computing model is composed of three main service models, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), capturing the capability of deploying, for customers from *fundamental computing resources*, to *libraries and tools/platforms* supported by the provider, and to provider's *applications*. The characteristics of cloud computing are presented in detail in Section 2. Probably the biggest benefit that cloud computing brings is that the services are bought *on-demand*. This means that, when a business needs infrastructure resources, or software platform and their licenses, for a limited amount of time, these can be bought on-demand, only for the time that they are needed. This is highly convenient for businesses, which have higher flexibility in terms of IT investments, and can provide their users expected Quality of Service, even for unexpected loads (e.g., sale periods that usually entail applications outages[1]). A Goldman Sachs study published in January 2015 reveals that, in the period 2013-2018, infrastructure and platforms are expected to grow at 30% compound annual growth rate, as opposed to overall enterprise IT which is expected to grow at 5% [45]. This growth is a response to the growing trend of moving applications in the cloud [50], or of developing the so-called *born-in-the-cloud services* [54].

Figure 1.1 shows the layers of cloud computing offerings, starting from the infrastructure and up to services. The platform layer can be built standalone (i.e., using

[1] `https://www.internetretailer.com/2015/03/11/\/-apples-itunes-and-app-stores-experience-outages-worldwide`

its own hardware resources), or it can use a private or a public IaaS cloud, in order to deploy the software/tools that should be offered as a service. Likewise, the service layer can be built standalone, or it could be based on IaaS or PaaS clouds. Since this is a service-based model, stakeholders play a central role. Since this is a hierarchical construction, customers of lower level layers can act as providers to upper layers (e.g., PaaS provider can act as customer to IaaS provider, and provider to SaaS provider). Another type of stakeholder is the customer to the SaaS provider, that is oblivious to whether or not the service s/he is using is using cloud services. In this kind of scenario, it can be quite an effort, for SaaS provider or PaaS providers to manage their services, respectively platforms, and to offer them at the desired quality under certain cost limits.

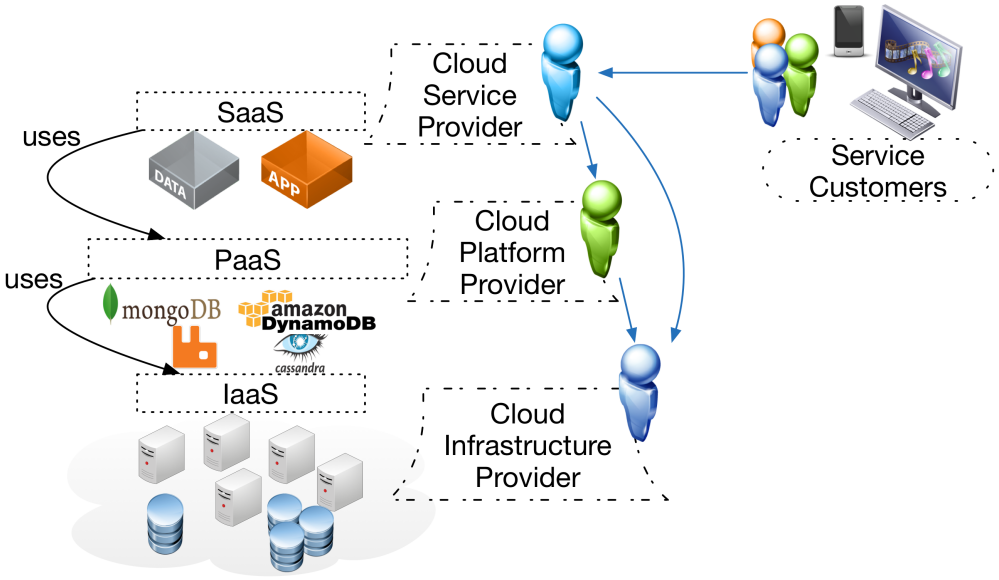


Figure 1.1: Service models and stakeholders in cloud computing

1.1 Problem Statement

In this context, we can see that when application owners would like to deploy their applications in the cloud and offer them as a service, when trying to use IaaS or PaaS cloud-offered services, they would need to hire cloud computing consultants or organizations specialized in this kind of development and operations activities. For deploying an application in the cloud, one needs to understand which cloud to use, what the available resources are, and which resources are best for the application at hand, given the budget allocated for hosting it in the cloud.

Once this step is completed, *at runtime*, application owners, who have now become service providers, need to understand how to manage their new service[2] over IaaS or PaaS

[2]We refer from hereon to any service, application, or system, deployed on a single or multiple cloud

environments. Focusing on this run-time perspective, we can see that it is very important how the application behaves, since that determines the Quality of Service offered to service customers. This behavior is affected by the load (e.g., the number of customers currently active and the operations they perform), and by the cloud-offered services that are currently used (e.g., how many VMs are used, which is the network performance, how much storage is available). With current state of the art management and monitoring tools (e.g., Amazon AutoScale[3], Google Autoscaler[4] or Google Cloud Monitoring[5]), the service provider gets reports on resource usage, and needs to decide whether further resources are necessary. Therefore, the service provider should understand how resource usage affects application-level metrics (e.g., response time), and specify rules for adding/removing IaaS/PaaS services. This *specification process* can be quite cumbersome, since it implies writing numerous "if-then-else" style policies for all resource-level metrics. Moreover, in the case of complex services, whoever is managing the service should understand the impact of his/her actions upon various service parts. For instance, when removing a node from a database cluster, the result might be of decreasing throughput in the service business end component.

One of the key arguments for cloud computing is the property of elasticity [35], that is, the ability of cloud services to acquire and release cloud-offered services on-demand, in response to run-time fluctuating workloads. For achieving such elasticity, services need automated controllers that know when new cloud-offered services (e.g., VMs, disks, or software platforms) are needed, and how to contact respective cloud provider in order to purchase them. Although currently cloud providers offer possibilities for customers to control their applications (e.g., Amazon AutoScale[3], or Google Autoscaler[4]), they consider only system-level metrics (e.g., CPU usage, or memory usage). Existing research on cloud services control focuses on providing best trade-off for users, between cost and resource usage [63, 94, 120]. However, *controlling the elasticity of complex services* implies understanding the service structure, the connections existent among different service parts, the artifacts necessary to be controlled for each service part, and the application specific metrics that reflect better the behavior of the cloud service.

1.2 Research Questions

The problems described in Section 1.3 motivate the research conducted as part of this thesis. More concretely, the following questions are researched in this work.

Research Question I
How could requirements be specified, with the least effort from service provider side, but still giving sufficient information for controlling the service?

infrastructures as *cloud service*, and to any service offered by cloud providers (i.e., IaaS, PaaS, and SaaS), cloud-offered service.

[3] http://aws.amazon.com/autoscaling/

[4] https://cloud.google.com/compute/docs/autoscaler/

[5] https://cloud.google.com/monitoring/

As discussed in Section 1.3, the current ways of specifying requirements are very difficult to be used by service provider, since they entail detailed resource requirements specification, and feature very limited abstractions. The service provider needs to be able to specify details that are relevant for him/her, regarding service performance, or service cost. Moreover, when the service has a very complex structure, the service provider might need to specify requirements on the different parts of the service. Although some research work, discussed in Chapter 10, exists on elasticity requirements specification, it doesn't consider service complexity, and mostly follows the mechanisms offered by cloud providers.

Research Question II

How can we control the elasticity of complex services deployed in the cloud?

The challenge here is controlling the cloud service considering high-level elasticity requirements, described above. As described in Section 1.3, most research focuses on controlling very specific service types (e.g., web applications [63], workflows [81]), and doesn't consider the subjective requirements when it comes to performance and cost (e.g., the current budget that the service provider allocates per service customer might be less then optimal when it comes to the overall performance, but s/he is not willing to pay more than that). Moreover, for controlling the service one needs to understand the service behavior, i.e., how various control actions affect, *in time*, different service parts.

Research Question III

How could elasticity control be integrated in service operations management?

The third research question assumes the previous two research questions are already solved. In this case, let us consider real-life services, which are provided by organizations with multiple employees, each with their own responsibilities. Normally, these organizations follow IT Service Management standards or best practices (e.g., ITIL [10]). Therefore, it needs to be investigated, which is the role the automated elasticity controller plays within the organization, and how are normal employees interactions affected by the fact that the service is running in the cloud, and it is controlled at runtime by automated software.

1.3 Scientific Contributions

In what follows we briefly present the scientific contributions made to the state of the art, in our quest of solving research questions stated in Section 1.2.

Contribution I

An elasticity requirements specification language

As discussed in Section , given the business perspective in cloud service elasticity, service providers could have different requirements, even for the same service type,

considering their business strategies and expected customers. Moreover, both determining and specifying resource-level requirements can be quite cumbersome. For this, we introduced SYBL (Simple Yet Beautiful Language) elasticity requirements specification language, which enables the specification of high-level elasticity requirements, at multiple levels of abstraction. The language is currently used in CAMF[6] Eclipse plugin, which enables the description and deployment of applications over various cloud providers. Contribution I was originally presented in [25].

Contribution II
A framework for controlling service elasticity

For fulfilling the requirements specified through the SYBL language, we introduced a model in order to be able to represent structural information, runtime information, and elasticity information concerning the service. A framework, rSYBL, which is based on this model and new elasticity control mechanisms were introduced, for being able to control service elasticity at multiple levels of abstraction. Contribution II was originally presented in [24].

Contribution III
Mechanisms for determining cloud service elasticity behavior

For accurately controlling cloud service elasticity, one needs to understand the behavior of different parts of the service, and the impact of various actions upon various parts of the service. Moreover, it is important understanding whether or not an enforced action would result in requirements violations, even for short periods of time (e.g., until the system stabilizes). For this, we have introduced a clustering-based approach, and a corresponding framework (ADVISE), which is able to estimate the behavior, in time, of different parts of the service, not only of the one on which the action is enforced. Contribution III was originally presented in [27], and awarded "best paper award".

Contribution IV
Mechanisms for controlling cloud service elasticity in heterogeneous clouds

When increasing the support for cloud services, we observed that IoT services need to run across multiple clouds (i.e., the so called cloudlets or mini-clouds and public/private clouds). Therefore, the controller needs to support the heterogeneity of various APIs through which virtual resources should be controlled, and understand relationships among these clouds. In this sense, rSYBL has been extended, for supporting heterogeneous enforcement mechanisms, by enriching the model it is based on and adapting the control algorithm for taking into consideration how control in one cloud might affect the parts of

[6] http://linc.ucy.ac.cy/CAMF/
[6] tuwiendsg.github.io/rSYBL
[6] tuwiendsg.github.io/ADVISE

the service deployed in a different cloud. Contribution |V was originally presented in [26]

Contribution V

A framework for cloud service operations management for elasticity

Although above contributions do ease controlling services in cloud-based environments, when considering large organizations, following various standards and having fixed processes in-place, using this kind of controller seems rather ad-hoc. As opposed to having a single organization employee using the elasticity controller and discussing with other employees regarding the desired performance, the desired cost, or the possible changes which may appear (e.g., cloud provider changing its schema), we proposed integrating the elasticity controller in organization's processes for IT Service Operation Management. An elasticity Operations Management Framework (eOMP) was designed and implemented, supporting interactions between the elasticity controller and various organization employees according to their responsibilities and authority. Contribution V was originally presented in [28].

1.4 Thesis Organization

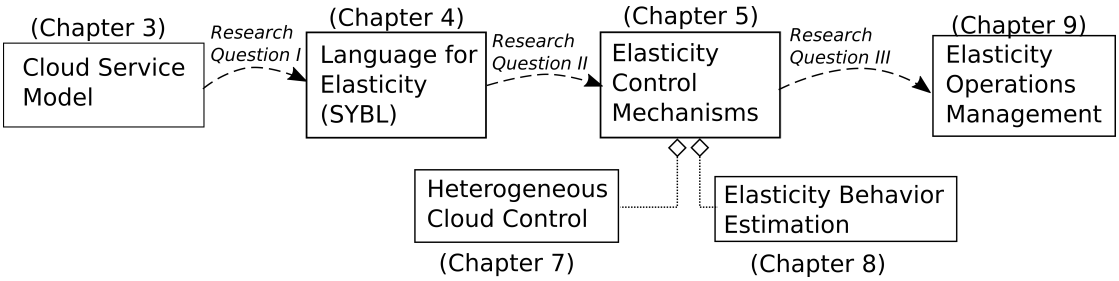


Figure 1.2: Thesis flow

Figure 1.2 gives an insight into the flow of this thesis. The remainder of this thesis is organized as follows:

- Chapter 2 provides context for this thesis, describing background information on concepts and techniques used.
- Chapter 3 discusses how the research questions and the techniques used in this thesis relate to schools of thought in science.
- Chapter 4 presents the model of the service, and the case study which will be used (partially or completely) for exemplifying concepts or for evaluation purposes throughout this thesis.
- Chapter 5 presents the elasticity requirements specification language

- Chapter 6 introduces the rSYBL framework, and the elasticity control mechanisms which the framework is based on.
- Chapter 7 describes the approach used for estimating elasticity behavior of the cloud service
- Chapter 8 presents the mechanisms introduced for supporting heterogeneous control for cloud services
- Chapter 9 introduces the elasticity Operations Management framework, showing how various employees with various responsibilities can interact with the elasticity controller and can collaborate for customizing and supervising the elasticity control in real-time.
- Chapter 10 concludes this thesis, and discusses future work based on the work performed as part of the thesis.

CHAPTER 2

Background

In this chapter we present basic concepts that will be used in the thesis, as well as existing tools/research that were used. We first describe the cloud computing models, focusing on the service models it is composed of, on stakeholders that interact in this context, and on current cloud providers' offerings. We present MELA [89], a tool for cloud service monitoring and analysis, which is being used by rSYBL elasticity controller presented in this thesis. Next, we present an overview of control mechanisms and machine learning techniques, which were used for rSYBL. The last section of this chapter presents concepts from IT Service Operation Management, which were used for Contribution V (see Section 1.3).

2.1 Cloud Computing

In recent years, cloud computing has been present in our lives at various levels: phones send video recordings or images "on the cloud", most applications, be they mobile or desktop, save user data "on the cloud", even banks are moving towards cloud [1]. In this section we are focusing on explaining what cloud computing is, detailing its model, the stakeholders collaborate in the context of cloud computing, and cloud standards. Next, tools used throughout this thesis are introduced.

Armbrust et al. describe the cloud in their 2009 vision paper [46] as having, the following novelty aspects: the illusion of infinite computing resources to be offered, the elimination of an up-front commitment by customers, and the ability of paying for use for computing resources or platforms which are used.

NIST defines the cloud as having five essential characteristics [86]:

- on-demand self-service - the cloud consumers can consume their services automatically, without needing to interact with cloud provider employees

[1] http://www.forbes.com/sites/tomgroenfeldt/2014/06/26/some-banks-are-heading-to-the-cloud-more-are-planning-to/

- broad network access - the computing resources or platforms can be accessed from any device connected to the Internet
- resource pooling - the resources (e.g., virtual processor, storage, network resourced) are pooled and managed by the cloud provider
- rapid elasticity - the capacity of the used virtual resources can grow and shrink rapidly. This way the self-service and resource pooling are enablers for rapid elasticity, managed from the customer size.
- measured service - the resource usage is monitored and reported by cloud providers.

2.1.1 Cloud Service and Deployment Models

The virtual resources are offered in the cloud according to three different service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). As shown in Figure 1.1, the three are hierarchically build (i.e., one can provide service to another one). In the IaaS model, the provider can provision "processing, storage, networks, and other fundamental computing resources" [86]. On these, the consumer is able to deploy and run arbitrary software. In the PaaS model, PaaS consumers can deploy their own applications "using programming languages, libraries, services and tools supported by the provider" [86]. In the SaaS model, SaaS consumers can use "provider's applications running on a cloud infrastructure" [86]. Guided by these models, and based on cloud computing, several initiatives appeared of offering everything as a service (XaaS) (e.g., sensing as a service [132] [104], everything as a service [11]).

2.1.2 Cloud Standards

Given the novelty of cloud computing, the standards in this domain are still evolving, and new ones are appearing. Moreover, given cloud's popularity, an abundance of organizations, and multiple sub-groups, have been interested in defining standards or best practices, each from a different perspective[2]: Distributed Management Task Force (DMTF), European Telecomunications Standards Institute (ETSI), Global Inter-Cloud Technology Forum (GICTF), Open Grid Forum (OGF), Object Management Group (OMG), Open Cloud Consortium (OCC), Organization for the Advancement of Structured Information Standards (OASIS), Storage Networking Industry Association (SNIA), Cloud Work Group (CWG), Association for Retail Technology Standards (ARTS), TM Forum, and OpenCloud Connect. The cloud hype has created an abundance of standards and open source activity, which has also lead to market confusion. To address this, the Cloud Standards Customer Council (CSCC), an end user advocacy group with the goal of accelerating cloud's successful adoption, oriented towards the customer. CSCC has issued a series of documents[3] describing best practices in cloud computing, and roadmaps for cloud security, or for migrating applications to the cloud.

[2] http://cloud-standards.org/

[3] http://cloud-council.org/resource-hub.htm

Cloud Service Description

Open Virtualization Format

The Open Virtualization Format (OVF) [100] was introduced by Distributed Management Task Force (DMTF), and is an open-source standard for packaging and distributing software and applications for virtual machines (VM). The standard, when proposed in 2007 by VMware, Dell, HP, IBM, Microsoft and XenSource, was initially intended for virtual machines. An OVF package consists of a folder containing several packages and artifacts, and an XML descriptor, which describes the packaged virtual machine.

Cloud Infrastructure Management Interface

The Cloud Infrastructure Management Interface (CIMI) [32] was introduced by Distributed Management Task Force (DMTF), and is an open standard for managing cloud infrastructure. CIMI standardizes interactions between cloud environments to achieve interoperable cloud infrastructure management. Although it has the advantage of clear specifications and of being a very concrete standard, the drawback is that unique services offered only by some cloud providers are simply ignored. However, CIMI has clear templates (e.g., for provisioning new resources such as machines, volumes, or networks) and interaction protocols (e.g., it even offers REST services description), having several implementations (e.g., Apache DeltaCloud[4], OW2 Sirroco[5], or StratusLab[6]).

Open Cloud Computing Interface

The Open Cloud Computing Interface (OCCI) is a set of specifications delivered by the Open Grid Forum, for cloud computing service providers. OCCI is more generic than CIMI, providing a skeleton on the basis of which one can extend it even for Platform as a Service. There are multiple implementations for OCCI (Ruby-based rOCCI[7], or Python-based pySSF[8]), or projects where OCCI was used for modeling the services (OpenNebula [9], CloudStack[10], or European Grid Infrastructure[11]).

Cloud Application Management for Platforms

The Cloud Application Management for Platforms (CAMP) [18] proposed by OASIS is intended to provide descriptions for PaaS cloud services. They propose a protocol for cloud customers to package and deploy their applications, defining interfaces for provisioning, monitoring and control. In this specification, the details of the infrastructure are hidden from the consumer, which only needs to define its artifacts and specify which

[4] https://deltacloud.apache.org/
[5] http://wiki.sirocco.ow2.org/
[6] http://stratuslab.org/
[7] http://github.com/gwdg/rOCCI
[8] https://github.com/tmetsch/pyssf
[9] http://opennebula.org/
[10] https://cloudstack.apache.org/
[11] http://www.egi.eu/

are the services used from the provider offerings. Implementations of CAMP include Project Solum[12] and Brooklyn[13].

Topology and Orchestration Specification for Cloud Applications

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [98] is an OASIS standard meant for large-scale applications, for which the cloud customer needs to describe multiple artifacts and complex relationships among them. TOSCA contains a series of features, like policies specification, or specification of complex action plans which can be described as process models, described using BPMN terminology. However, TOSCA is not yet widely used commercially, since vendors have been developing their own specifications, in parallel with the TOSCA standard (e.g., OpenStack Heat[14] or Amazon AWS CloudFormation Template[15]).

Cloud Computing Reference Architectures

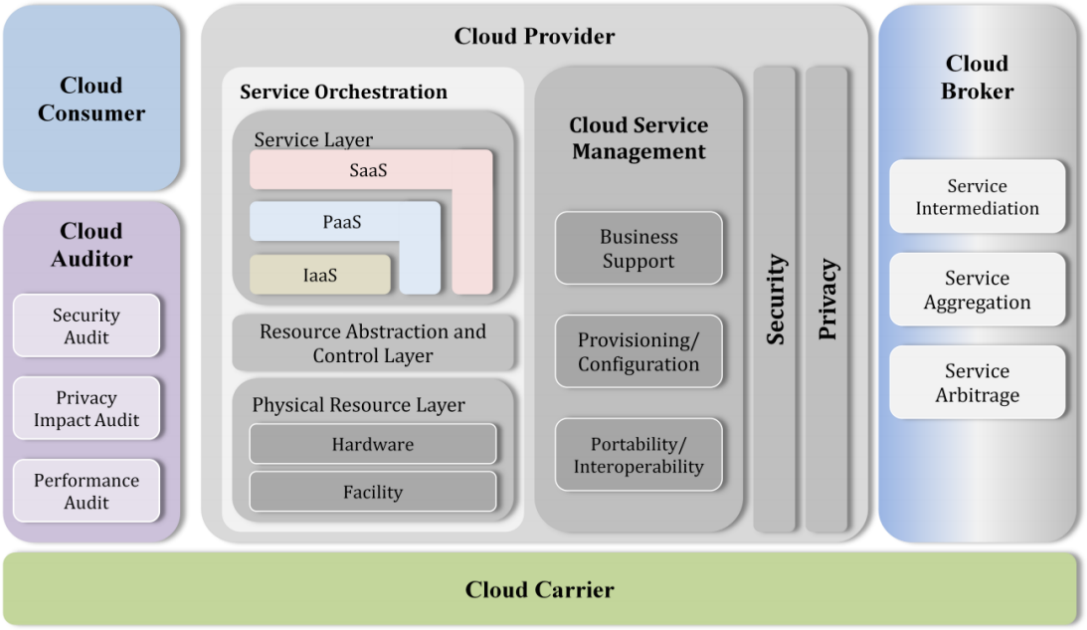
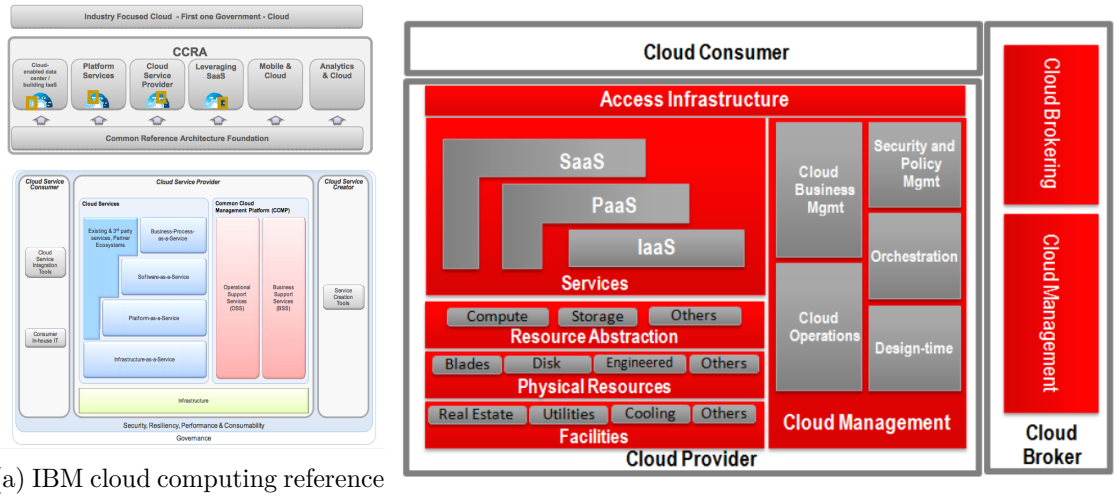


Figure 2.1: NIST cloud computing reference architecture overview

Several cloud computing reference architectures exist, e.g., NIST Cloud Computing Reference Architecture [96], IBM Cloud Computing Reference Architecture [61], or Oracle Cloud Computing Reference Architecture [99]. The NIST architecture, shown in 2.1 is the

[12] http://solum.io/
[13] https://brooklyn.incubator.apache.org/
[14] http://docs.openstack.org/developer/heat/
[15] http://aws.amazon.com/cloudformation/aws-cloudformation-templates/

most used, it being initially defined in 2011. The architectures model the cloud services, and cloud computing main actors. NIST defines the following action types: (i) *Cloud Consumer*, which is a person using services from Cloud Providers, (ii) *Cloud Provider*, a person making the service available to interested parties, (iii) *Cloud Auditor*, which is an entity that can conduct independent assessment of cloud services, infrastructure systems operations and security of the cloud implementation, (iv) *Cloud Broker*, that manages the interaction in terms of performance and delivery of cloud services, also negotiating relationships between *Cloud Providers* and *Cloud Consumers* and (v) *Cloud Carrier*, an intermediary providing connectivity between *Cloud Providers* and *Cloud Consumers* (e.g., Internet provider).



(a) IBM cloud computing reference architecture overview

(b) Oracle cloud computing reference architecture overview

Figure 2.2

IBM cloud computing reference architecture (see Figure 2.2a) adds to the cloud computing traditional service models *Business-Process-as-a-Service*, and two verticals for *Common Cloud Management Platform*, *Operational Support Services* and *Business Support Services*. This architecture contains only three types of actors, *Cloud Service Creator*, who is creating the service leveraging functionality offered by *Cloud Service Provider*, *Cloud Service Provider* who owns a common cloud management platform, and has the responsibility of providing cloud services, and *Cloud Service Consumer*, who uses cloud service integration tools and consumer in-house IT in order to use the cloud service.

Oracle cloud computing reference architecture (see Figure 2.2b) is more detailed in terms of connections existent among IaaS, SaaS and PaaS, introducing *Cloud Builder* that builds and operates the cloud infrastructure and platforms as a service, *Cloud Application Builder(s)*, which develop applications for the cloud, and deploy them on the PaaS platform and offer as SaaS services, and *Application Management* which includes self-service capabilities to provision and manage applications deployed in the cloud. More importantly, the Oracle cloud computing reference architecture contains contracts,

binding agreements between Cloud Builder, Cloud Application Builder or SaaS consumer.

2.1.3 Cloud Tools

In this thesis several tools were used, in order to be able to understand the service, and develop the techniques that were mentioned as scientific contribution in Chapter 1.3. For monitoring and analyzing the service behavior, MELA [89] was used, while for enforcing control strategies on the cloud service, several tools were used like Salsa [73], JClouds[16], Flexiant Cloud Orchestrator (FCO)[17].

MELA: Cloud Service Monitoring and Analysis

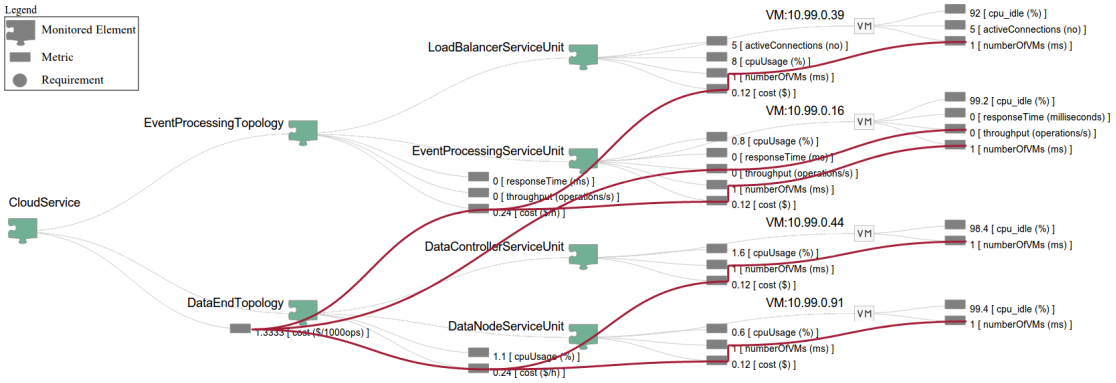


Figure 2.3: MELA snapshot on structuring monitoring information

For understanding the service behavior, one needs to monitor it. Several solutions, like Ganglia[18], Nagios[19], or JCatascopia [119], facilitate monitoring resource usage, and expose various plug-ins mechanisms for adding different types of metrics. However, the virtual resources that these tools are monitoring are volatile, and it is quite difficult to understand the meaning that a value of a metric for a single resource has on the overall performance of the application.

MELA is a mature tool for monitoring and analysis of elastic cloud services[20], structuring and enriching information collected from existing monitoring solutions (e.g., Ganglia, JCatascopia). In MELA several concepts are introduced for representing the cloud service behavior analysis: *Elasticity Boundary*, *Elasticity Space*, and *Elasticity Pathway*. The Elasticity Boundary gives a metric's minimum and maximum values, for which all requirements were fulfilled. The Elasticity Space is the union of all boundaries for all cloud service's metrics. The Elasticity Pathway gives the evolution of the elasticity

[16] https://jclouds.apache.org/
[17] https://www.flexiant.com/flexiant-cloud-orchestrator/
[18] http://ganglia.sourceforge.net/
[19] https://www.nagios.org/
[20] http://tuwiendsg.github.io/MELA/

space over time. Figure 2.3 shows a snapshot from MELA user interface, structuring monitoring information on multiple levels of abstraction, and defining new metrics (e.g., cost per client per hour) considering the lower level ones.

2.2 Control Mechanisms

In computer science, various areas deal with adaptation/management/control of software, or hardware systems. For controlling cloud services, we look into techniques that facilitate the control of the service, while considering the problems stated in Section 1.3.

2.2.1 Closed-loop control

Control theory is in charge with controlling the behavior of dynamic systems. These systems are supposed to have an input, a way to interact with them for altering their behavior (i.e., enforcing actions on the system), and an output, which is a way of monitoring the system. There are two main types of control systems: (i) open-loop control system, and (ii) closed-loop control system. Open loop control systems compute their control strategies using only the current state and the model of the system, without having a feedback from the system. Closed loop control systems are also taking monitoring information from the system, thus closing the control loop. Figure 2.4 depicts a closed loop control, where the controller enforces, as system input, actions for modifying system's behavior. The system output is captured as a feedback, and sent to the controller to use it in the closed-loop transfer function for computing new actions to enforce.

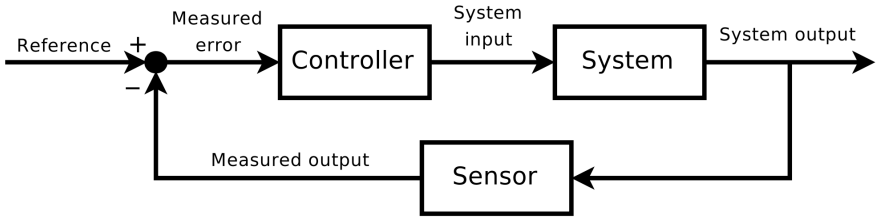


Figure 2.4: Generic feedback control

2.2.2 Levels of automation in control

Full automation is mostly possible for simple tasks. For controlling complex systems, humans are still considered playing various parts in the system control, depending on the *Level of Automation*. Throughout the years, the participation of humans to automated processes has been classified in various taxonomies.

When considering the ones described above for controlling complex systems, sometimes levels 10 from Sheridan taxonomy and 5 from Endsley taxonomy are not realizable. This is why, the automation levels immediately below are sometimes used. This type of control is referred in the literature as *supervisory control* [111]. In this model, the human

s

Taxonomy	**L**evel	**D**escription
Sheridan [113]	1	The computer offers no assistance, and humans must take all decisions
	2	The computer offers a complete set of possible decisions
	3	The computer offers a selection of possible decisions
	4	he computer suggests one single decision
	5	The computer executes a suggestion if the human approves it
	6	The computer allows humans to veto the decisions for a restricted time
	7	The computer executes the automatically the decision and informs the human
	8	The computer executes the decision and informs the human only if asked
	9	The computer executes the decision and decides whether it should inform the human
	10	The computer decides everything and acts autonomously, ignoring the human
Endsley [37]	1	The computer offers no assistance, humans should take all decisions and enforce all actions
	2	The computer offers recommendations, and the human takes the decisions
	3	Consensual artificial intelligence, with the consent of the human required to carry out actions
	4	Monitored artificial intelligence, with system autonomously implementing actions unless vetoed by humans
	5	Full automation, with no human interaction

Table 2.1: Levels of autonomy

supervisor monitors the behavior of the automation performing the task (i.e., controlling the system), and detects or is reported failures and abnormalities. In other words, supervisory control offers automation of all functions, with human override capability.

2.2.3 (Cloud) Requirements Specification: Service Level Agreement and Scaling Policies

Quality requirements coming from users or system owners normally guide any kind of management/adaptation/control process enforce upon the system. Quality of service traditionally refers to network quality, but is a term being applied also generally for quality of offered services/applications. Service level agreements are contracts among

service provider and service user, specifying the agreed characteristics of the service, like quality, costs, or responsibilities. In cloud computing, service level agreement doesn't play a central role in the cloud provider offerings, being quite shallowly described (e.g., just in terms of availability).

Several standards have been designed for SLA description, like Web Service Level Agreement [60]. For cloud computing SLA, effort has been put towards standardizing SLA description (e.g., white paper from EU on cloud SLA standardization [21]) with no concrete result up until now. WSLA comprises (i) a description of the parties, and their roles, (ii) a description of parameters/metrics and details on how to measure them, and (iii) service level objectives, which are guarantees of certain SLA states for particular periods. In cloud computing, cloud providers define generic agreements, to be used for all their services (e.g., AWS Agreement [22]).

For managing/controlling services, current cloud providers are using input from the users in the form of policies. For instance, in Google Autoscaler, the user can specify policies of scaling considering average CPU utilization, considering custom metrics or based on HTTP load balancing serving capacity[23]. Similarly, Amazon AutoScaling[24] facilitates the description of scaling policies depending on various custom metrics, but allows two types of scaling: simple scaling, with increasing/decreasing current capacity of the group based on single scaling adjustment, and step scaling, which increases/decreases current capacity based on a set of scaling adjustments, that vary based on the size of the alarm breach.

2.3 IT Service Management

IT Service Management is the process of planning and controlling the quality and quantity of provided IT services, and can be defined as *"a subset of Service Science that focuses on IT operations such as service delivery and service support"* [48]. Most known ITSM framework is IT Infrastructure Library (ITIL) [10], containing a set of well defined, generic, processes, procedures and tasks, for businesses. Other ITSM frameworks include COBIT [25], ISO/IEC 20000-1:2011 [17], or Microsoft Operations Framework[26].

Focusing on ITIL, it defines the following processes (see Figure 2.5) Service Strategy, Service Design, Service Transition and Service Operation. Since in this thesis we are focusing on run-time elasticity control, our focus from IT Service Management will be Operation Management.

[21] http://ec.europa.eu/digital-agenda/en/news/cloud-service-level-agreement-standardisation-guidelines

[22] http://aws.amazon.com/ec2/sla/

[23] https://cloud.google.com/compute/docs/autoscaler/

[24] http://aws.amazon.com/autoscaling/

[25] http://www.isaca.org/cobit/pages/default.aspx

[26] https://technet.microsoft.com/en-us/solutionaccelerators/dd320379.aspx

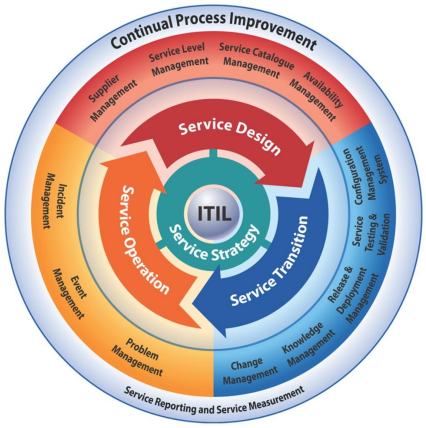


Figure 2.5: IT service management phases

2.3.1 Operation Management

The main objective of service operation is to carry out the activities and processes required to **deliver and manage services at agreed levels to business users and customers**[27]. Service operation processes include *Request Fulfillment*, *Incident Management*, *Problem Management*, *Access Management* and *Event Management*.

Request fulfillment processes have as scope dealing with requests coming from the users, assisting with general information, and complains. Incident management has as goal to restore service operation at normal functioning, and to minimize the impact of incidents on business operations. The problem management is in charge with minimizing the adverse impact of incidents on the business, and to prevent recurrence of incidents. Access management is in charge with protecting confidentiality, integrity and availability, and managing the way security incidents and problems related to access management are recorded. Event management provides the ability to detect, interpret and initiate appropriate actions for events, and provides operational information as well as warnings and exceptions to aid automation.

The following roles are more relevant to operation management[28], from the complete list of roles in ITSM [21]:

- *Access Manager* - grants rights to use the service, while preventing non-authorized users

- *Facilities Manager* - manages the physical environment where the IT infrastructure is located.

[27] https://www.ucisa.ac.uk/~/media/Files/members/activities/ITIL/service_operation/ITIL_Introducing%20Service%20Operation%20pdf.ashx

[28] http://wiki.en.it-processmaps.com/index.php/ITIL_Roles$#$ITIL_roles_and_boards_-_Service_Operation

- *Incident Manager* - responsible for the effective implementation of the Incident Management process and carries out the corresponding reporting.
- *IT Operations Manager* - takes overall responsibility for Service Operation activities.
- *IT Operator* - perform the day-to-day operational activities (e.g., backups, ensuring scheduled jobs are performed, installing new hardware/software in the data center).
- *Problem Manager* - responsible for managing the lifecycle of all problems, minimize the impact of incidents or even prevent them where possible.

CHAPTER 3

On Elasticity Control

This chapter is a reflection on the thesis topic in relation to two major schools of thought in philosophy of science: dialectics and constructivism. I gratefully thank Prof. Christiane Floyd for the most insightful lecture on philosophy of science, and the opportunity of writing this under her guidance at the beginning of my PhD studies.

3.1 Overview

This chapter presents an analysis of elasticity with the purpose of controlling cloud service elasticity, taking into account that a standard definition for elasticity has not yet been defined in cloud computing. We showcase a multi-perspective analysis of elasticity, and construct service elasticity as reality in the cloud, through language description. The detail description of the language can be found in Chapter 5. For the analysis, we take insights from two major schools of thought in philosophy of science: dialectics with main focus on the Scandinavian school of thought that promotes multiple perspectives in informatics, and constructivism.

3.1.1 Hegelian Dialectics and Multiple Perspectives from the Scandinavians

Hegelian dialectics proposes a dual model based on thesis and antithesis which emphasises discourse as a method of promoting synthesis. With relation to dialectics, multi-perspective reflection in science promotes mediating between different views of the same problem for achieving a better model. Multi-perspective reflection is presented by Nygaard [97] as one of the four main aspects of sciences, next to phenomenology, analysis and synthesis. Cloud computing can be considered as being part of informatics, business and even society, due to its social impact. We consider it however as being a sub-domain of informatics, but due to its many influences on other sciences, the types of stakeholders affected and their perspectives differ greatly. We adopt Nygaard's definition

for *perspectives* [97], seen as different world views that can coexist with each other. This definition contrasts with the paradigm definition for which Nygaard adopts Thomas Kuhn's view [14] of basic perspectives within a science, which are irreconcilable with others.

3.1.2 Constructivism

In the radical constructivist view of Ernst von Glaserfeld, the science and the cognition helps *with the organization of the experiential world, not the discovery of ontological reality* [51]. The constructivist view also promotes multiple perspectives in constructing the world, since a constructed world belongs to the individual, but through social interaction individual worlds form a consensual domain [52].

In the view of the constructivist Heinz von Foerster [44], scientific hypotheses are stories, invention of poets which compete for getting public's approval. We construct our own world and the validity of our world depends on the data we use: *The world, as we perceive it, is our own invention*[1].

We use a constructivist approach on elasticity understanding: analyzing different perspectives of the reality, in this case composed of stakeholders' views and current technologies, we choose characteristics for service elasticity construction, which will help in the definition of the new language.

3.2 Cloud Computing and Elasticity

Cloud computing is envisioned by Wang et al. [126] as providing user centric interfaces, on-demand service provisioning, QoS guaranteed offer and most importantly, ensuring scalability and flexibility. The enumerated properties were seen as main characteristics of cloud computing when this computing perspective appeared. However, the last two properties, scalability and flexibility, as well as QoS guaranteed offering are difficult to be ensured. Firstly, this is due to the fact that for the cloud provider (the stakeholder offering cloud services) it is challenging to know what scalability, flexibility and elasticity is for the cloud customer. Moreover, the cloud provider has to ensure maintaining quality properties while guaranteeing all the defining properties of cloud computing.

3.2.1 Multiple Perspectives of/for Cloud Computing

Cloud computing is a model with major interest not just for parties from the domain of informatics, but also from completely different sides like privacy, legal or business domains. In informatics, the interest mainly refers to defining and using cloud computing in the best possible way from the point of view of resources allocated, or performance. On the other hand, business stakeholders are interested in a different type of quality than the one informatics people refer to: quality of experience, as perceived by the user of their products that are hosted on the cloud. Moreover, the relation between the prices paid for

[1] `http://www.univie.ac.at/constructivism/HvF.htm`

cloud resources, and the quality obtained in return is essential to business stakeholders. This relation can be seen from the viewpoint of legal domain, posing several questions: (i) does the user get what he/she pays for? (ii) making a parallel to fuel market, how to regulate the cloud computing market? These are valid and important questions, but for finding answers, cloud computing professionals have to undergo several crucial steps. They firstly need to give cloud users the possibility of describing his/her requirements in terms of quality, cost, and resources, and not just in generic terms but also in correlation to the states of the service to be deployed on the cloud. Secondly, it needs to provide monitoring mechanisms in terms of quality, cost and resources, for a higher transparency both towards the user and towards the stakeholders supervising contract compliance (just as the money transfer can be monitored through the banks, the vice-versa transfer of quality and resources should be monitored). We therefore have multiple perspectives to consider in the cloud computing domain when contributing to its evolution.

3.2.2 Cloud Stakeholders

Language description implies a study of the users, in this case, cloud stakeholders, and their needs. A dialectic view towards cloud stakeholders would separate them into two: on one hand, the cloud customers and on the other hand, the cloud providers. This way of seeing the involved parties is rooted in the usual differences between the buyer and the seller: both want to give as little as possible while obtaining as much as possible. The parties have to discuss, debate, and most importantly, to negotiate a common agreed solution. In the recent years, numerous efforts have been put towards cloud service level agreement (SLA) negotiation and management [23, 123], but there is still plenty of room left.

Moreover, we may want to see cloud stakeholders from more perspectives, due to their complexity and due to the impact of this domain in business and on the society as whole. In cloud computing domain, a *stakeholder* is any user of services or services hosted on the cloud, to scientists promoting standards or even to investors in cloud provider companies. From different viewpoints, they are all interested in cloud computing, its evolution and performance. Marston et al. [84] give the following categories of stakeholders in cloud computing: (i) consumers (in other literature, cloud customers) are the subscribers who purchase the user of the cloud system, (ii) providers who own and operate the cloud computing systems, (iii) enablers who sell products that facilitate the cloud computing adoption and (iv) regulators who are objective stakeholders pervading across the other stakeholders and making sure that the agreed-upon contracts are fulfilled.

3.2.3 Elasticity in Cloud

Dustdar et al. [36] propose elastic processes, elasticity being a defining property of cloud computing which consists in the ability of a provider to manage resources allocated by scaling them up and down on an on-demand basis. However, it is practically impossible for the provider to guess how and when the service should be scaled, while respecting service security and without having any elasticity requirements specification from the

customer side. Elasticity specifications could describe the manner in which the hosted service should behave under different conditions (see Section 3.5 for a more detailed view). Several other service elasticity issues are beyond our current goal and can be encountered in the process of creating an elastic service, for instance how much can the service scale (i.e., if we allocate more resources, by horizontal or vertical scaling, will the service be able to fully make use of them?) or how to decide on workload distribution for the service.

3.3 Philosophical and Societal Views towards Requirements Specification and Engineering

Requirements description, analysis and engineering have had a long evolution to come to be part of the science of informatics. The Scandinavian school of thought played an important role in promoting the societal impact as a part of informatics. The impact the information systems have on the society is not yet fully part of the informatics science, but requirements engineering and methodologies for system design that consider user options are a great step forwards. Kristen Nygaard [97] proposed for the informatics science to also include the societal impact of programmers, which should consider all stakeholders perspectives when designing an information system. Holbaek-hanssen, Handlykken and Nygaard, part of the same Scandinavian school of thought, give an important description to information systems: *A system is a part of the world that a person (or group of persons, during some time interval and for some reason) chooses to regard as a whole consisting of* ***components****, each component characterized by* ***properties*** *that are selected as being relevant and by* ***actions*** *relating to these properties and those of other components* [57]. In other words, a system gets to be a system only if someone looking at it chooses the system perspective, by considering the world or current context as being composed of components. Otherwise, the current context can be described as a business environment, a household, etc. Therefore, the perspective through which we get to look at a context is definitive for understanding and describing it. Moreover, considering numerous perspectives when designing a model (system, component, computing artifact, etc.) will enrich both the model capabilities, and the situations in which it can be used. Just as for the definition of a system, language design needs to consider all stakeholders' perspectives and to meet all their needs. A more detailed discussion on stakeholders interested in service elasticity, their perspectives and needs will follow in Section 3.5.

Defining/introducing a language through which one can describe service elasticity gives the limits on which the elastic service can exist. This idea has one of its roots in Christiane Floyd's work regarding software development as reality construction [43]. Christiane Floyd adopts Hanz von Foester's perspective on reality as community, and sets the dialogue as the basis for any construction work, from dialogue resulting the adoption of other's perspectives. Software development is seen as the design of a world that links the social world of services to be developed, with the technical world of service implementation. This design entails a series of decisions from the perspective we take for considering requirements to the methods we choose to apply.

3.4 Ethics and Quality in the Cloud

3.4.1 Elasticity as a Property Promoting Quality

Service elasticity has been so far presumed as true, simply from the fact that the service was being deployed on the cloud. This is unfortunately not true, since elasticity implies the property of automatic scalability according to the service's reality. Service reality differs depending on the stakeholder describing the specific service. In other words, it is subjective stakeholder's perspective.

Specifying what elasticity entails, and what are the mechanisms of obtaining this property, is a defining step towards obtaining quality in cloud computing. It is difficult both for the cloud customer and the cloud provider to choose fixed performance, resource and cost numeric values, and stick to them. For the cloud provider, it is impossible to keep these metrics at a fixed level, while for the cloud customer it is extremely difficult to specify them since the metrics depend greatly on the workload. Moreover, these metrics and their sub-metrics (e.g., cost has as sub-metric the cost per IO). Clearly defining what are the desired characteristics of the service, and how it should behave under predefined condition, would enable keeping the service at expected quality for all the involved stakeholders.

3.4.2 Ethics in the Cloud - (EaaS a.k.a. Ethics as a Service?)

Cloud computing is intended to be the domain which links society with the information domain. Paying for services, resources, etc., in an on-demand basis, easily accessible by anyone, with no need of informatics knowledge, would have seemed several years ago as a sci-fi story. One of the main problems with this ideal world is that people do not trust it yet, and this is mainly due to the fact that there are no regulation entities, that can guarantee regular users that they will get what they pay for. This can be seen as a domain-specific problem, but also as a legal issue or ethics problem.

We therefore encounter several issues which lead to this lack of trust: (i) the cloud users have no mechanism of intuitively specifying their requirements in terms of service behavior (service elasticity) over time, (ii) there is no cloud authority to mediate these real-world transactions that exchange money for services, (iii) the pricing schemes that the cloud provider use are unknown to the common users - users do not know what they are actually charged for, they usually just receive the bill.

Enabling the customers to describe their requirements through a language can help promoting ethics and quality in the cloud, by enabling stakeholders to specify what should be considered about their service, what should be monitored and what are the mechanisms that they consider necessary for keeping the service in a "good" state both from the quality and cost perspective. It also helps constructing a new "reality", a new way of seeing elastic services, by merging views of different stakeholder types involved and projecting it to a specification language.

3.5 Cloud Service Elasticity: Perspectives

Cloud service automatic scalability is a property towards which cloud community evolved with small steps, from the initial cloud proposal [86] that proposed *rapid elasticity*.

Elasticity in the cloud is an essential property, defined by stakeholders' decisions from design to operation phases. This thesis provides an insight into how we control services, with focus on stakeholders' expectations and knowledge. As we will detail in Chapter 5, we look on how the service behaves and can be controlled from a multi-dimensional perspective: cost, quality and resources elasticity [36]. This means that we are not interested only to optimize cost with respect to service performance, but on a personalized control, depending on the needs of each stakeholder. Different stakeholders interested in service elasticity have views which depend on the stakeholder type and his/her interests (e.g., a cloud provider is usually interested in how resources are scaled for providing the promised quality while a cloud customer can be interested in the relation between cost elasticity and quality elasticity). Moreover, for controlling complex services, the control strategy might differ for different parts of the service (e.g., if the data cluster holds huge amounts of data, and is the most important part of the service, its cost can be very high when compared to frontend). When dealing with many stakeholders in charge directly (e.g., service manager, system administrator) or indirectly (e.g., cloud provider) of the service their service behavior goals need to be reconciled, and the service has to be controlled towards a common desiderate.

With focus on this multi-perspective view, this thesis offers an insight into challenges that appear when controlling cloud services, and proposes mechanisms, languages, and tools for addressing them.

CHAPTER 4

Model and Case Study

The focus in this chapter is firstly modeling the significant information that characterizes a cloud service, and secondly the case study that will be used throughout this thesis. The information that is important for describing cloud services is analyzed, and a model is introduced for being able to consistently represent this information. Furthermore, the case study application is described. The whole case study, or parts of it, is used throughout the thesis for motivating or evaluating presented work.

4.1 Cloud Service Model

4.1.1 Service Units

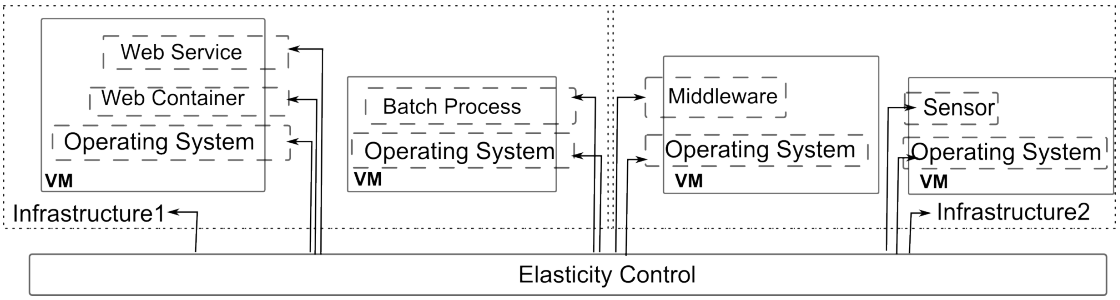


Figure 4.1: Emerging cloud services control

Many types of scientific, enterprise and government cloud services have been emerging [6, 62], which mix a series of component types, e.g., Machine-to-Machine (M2M) sensors, Web services/containers, and middleware. As shown in Figure 4.1, we can have, conceptually, a multitude of components running in the cloud, each with various capabilities. In cloud technologies, on the one hand, each of these components can be

re-configured during runtime. On the other hand, the cloud infrastructure consists of computing resources that are executed and have associated capabilities for creating/modifying them. These capabilities are offered at runtime as a service, and thus we call them "service units" [115].

Currently, most cloud control techniques scale only horizontally and at resource level the service unit (e.g., adding a new VM with the whole stack). However, understanding service units and their capabilities entails a highly granular control, using various types of control actions (e.g., change distribution mechanism for load balancing, change heap size, or change version), and combinations among them. These control actions can facilitate the fulfillment of a high range of requirements desired by cloud service stakeholders.

Finding the needed configurations for each situation, each artifact, while considering the complexity of the entire service is of utter importance for ensuring that we have an elastic cloud service during runtime. These configurations highly depend on the subjective requirements of the stakeholder for the service s/he has deployed, on the complexity of the service, and on the offerings of the cloud providers in terms of both software and infrastructure resources.

4.1.2 Cloud Service Structure

For specifying elasticity requirements at different levels, and then controlling elasticity at multiple levels we need to know the structure and particularities of the cloud service. Current cloud service specification standards like TOSCA [98] and CIMI [32] facilitate the service description prior to the deployment, the description containing all the information needed for the deployment process. However, as the purpose of these languages is not to describe the cloud service runtime behavior, they cannot describe mechanisms to achieve elasticity at the different levels. In order to generate and enforce control decisions during runtime, an elasticity controller would need to understand multiple types of information, e.g., information regarding cloud service units and the relation among them, information on the virtual resources used, or information regarding the cloud service developer/provider requirements. Therefore, we develop a representation model for our cloud service control, which overcomes the above-mentioned issues, and uses concepts proposed in the mentioned standards, like *Service Topology* or *Cloud Service*. This section discusses how various metrics and capabilities are associated to different parts of the cloud service or of the cloud infrastructure, and proposes a model for cloud service representation, which at runtime has the form of a dependency graph.

The cloud service description shown in Figure 4.2 is designed to provide to a cloud service controller with support for managing the cloud service. It holds different types of information: (i) structural/static information, (ii) virtual infrastructure related information, and (iii) elasticity related information. The cloud service can be seen as a graph composed of all this information, where each of the above concepts are nodes of the graph, descriptive information regarding the concept being modeled as node attributes and the relationships among them as edges connecting the various nodes.

The structural information describes the logical units out of which the cloud service is composed, and the relations between them:

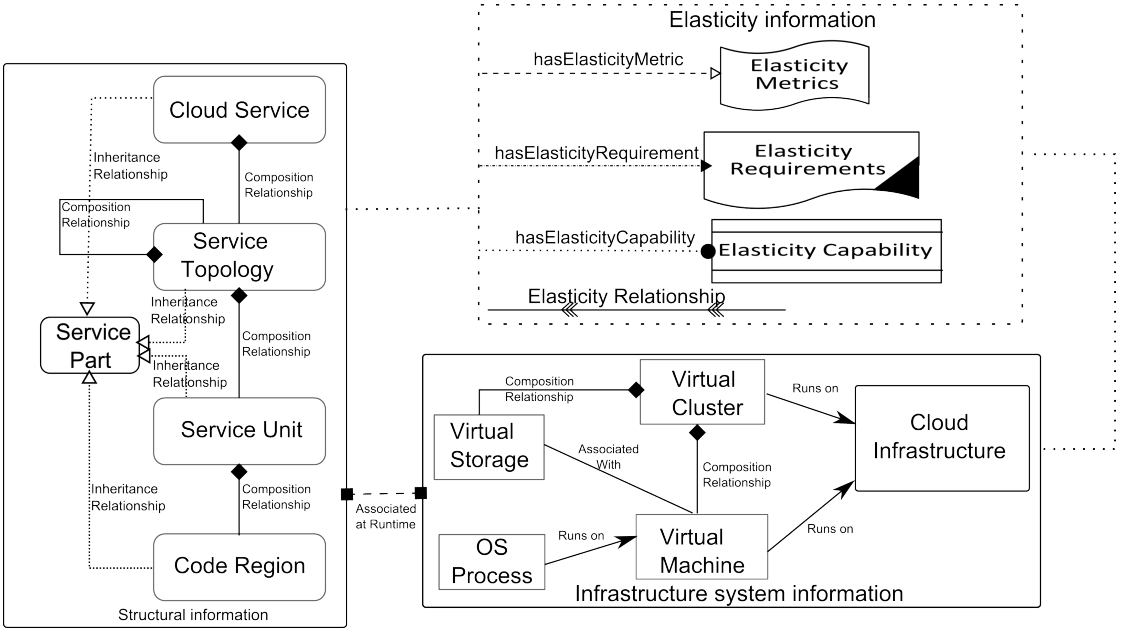


Figure 4.2: Linking structural, elasticity and infrastructure system information

- The *Cloud Service* represents the entire application or system, and can be further decomposed into service topologies and service units (e.g., a game, a web application, or a scientific application). The term cloud service that we choose to use is in accordance with existent cloud service architectures and standards (e.g., IBM Cloud Computing Reference Architecture [61] and TOSCA [98]).

- The *Service Unit* [115] represents any kind of artifact, component or service offering computation and data capabilities (e.g., a web service, or a data analysis service).

- The *Service Topology* represents a logical grouping of service units that are semantically connected and that have elasticity capabilities as a group (e.g., a tier of a cloud service, or a part of a workflow).

- The *Code Region* represents a particular sequence of code for which the user can have elasticity requirements (e.g., a data analytics algorithm).

The infrastructure related information enables the elasticity controller to be aware which unit is deployed on which VM, or which cloud provider:

- *OS Processes* represent any kind of processes belonging to a cloud service that can be associated either with code regions or with service units (e.g., a web server process).

- *Virtual Machine (VM)* and *Virtual Storage* are any IaaS services of type virtual machine and respectively storage that are purchased from the IaaS provider.

- The *Virtual Cluster* is a grouping of virtual machines or storage which have different properties (e.g., availability zone), and is offered as a service by the cloud provider.

This information regarding the infrastructure on which the cloud service is running is important in deciding how to control the service, since many of the actions depend on what the cloud provider offers. The above concepts (e.g., OS processes, or virtual cluster) are used to describe virtual resources in different cloud infrastructures[1]. This information regarding the infrastructure on which the cloud service is running is important in deciding how to control the service, since many of the actions depend on what the cloud provider offers.

The elasticity-related information facilitates the description of elasticity behavior for service units, service topologies or entire cloud service:

- *Elasticity Metrics* represent metrics targeted by elasticity requirements or lower-level metrics that are used for computing targeted metrics (e.g., cost vs. performance, cost vs. throughput, or cost vs. availability). Elasticity metrics can be associated with any cloud service part (e.g., service unit, service topology, or code region).

- *Elasticity Requirement*, represents any request coming from the user regarding elasticity of the cloud service (e.g., "the cost should not increase by more than 20% when the performance increases by less than 5%"). These requirements can be specified through SYBL and can be associated with any cloud service part.

- *Elasticity Capability*, represents any action/ mechanism/ operation through which the elasticity of the cloud service, of the service topology or of service units can be manipulated (e.g., the elastic reconfiguration of the data service topology for higher availability, or the elastic creation of new processing jobs for a map-reduce application).

- *Elasticity Relationship*, represents any connection between any two cloud service parts, which can be annotated with elasticity requirements (e.g., the connection between two service units needs to be of high reliability). We choose using the *relationship* term for being in accordance with cloud service specification standards (e.g., TOSCA).

We populate the graph constructed according to the model presented above with information from different sources (e.g., information from cloud providers regarding cloud infrastructure, pre-deployment information such as TOSCA description, or post-deployment associations between the static description and the virtual cloud infrastructure). Therefore, we do not assume that stakeholders will provide complete information at all the levels of the cloud service.

[1]Even though some names might differ, the actual concepts present high degree of similarity. E.g., Flexiant[2] uses `Server` for referring to VMs, offering `Disks` on the storage side, while Google Compute Engine[0] is offering various types of `Instances` (VMs), and `Storage`)

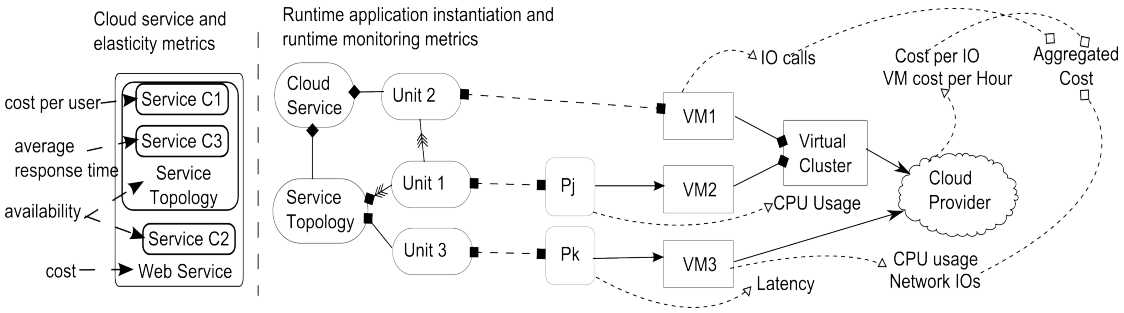


Figure 4.3: Constructing runtime dependency graph

4.1.3 Runtime Dependency Graph of Elastic Cloud Services

In order to describe the cloud service during runtime, a runtime elastic dependency graph is used. This dynamic graph captures all the information about the abstract model and runtime information like elasticity metrics, requirements and deployment topology during runtime and is constructed by the control system. Figure 4.3 shows how the runtime dependency graph is constructed. If we take the example of a Web service (the left side of Figure 4.3), the cloud user views his/her Web service as a set of services (in this case Service C1, Service C2, and Service C3), some of them grouped together for monitoring purposes (in this case Service Group which consists of Service C1 and Service C3). The metrics targeted in userâĂŹs elasticity requirements in this stage are high level metrics, referring to the quality, cost and resources of services, of groups of services or even of the entire Web service.

At runtime, the dependency graph is constructed (right part of the figure). Service instances are deployed on virtual machines, in different virtual clusters or even different cloud providers, being viewed by the runtime control system through the light of our model (e.g., Service C1 becomes Service Unit1, possibly having more than one instances deployed on more than one virtual machines), and the accessible metrics are low level ones. For the system to control such service it needs to know how to aggregate metrics for obtaining the higher level ones which are targeted by the user, and how services are linked together for having the capacity to properly control them. For bridging this information gap between the user and runtime perspective we use the dependency graph, which facilitates the control system of the cloud service to take control actions considering the complete description of the cloud service.

The runtime dependency graph is associated with metrics information from two views. Firstly, it is associated with metrics from the service user perspective that can view the cloud service as a Web service and has a high level view concerning metrics. Next, the graph is associated with metrics from the control system perspective which views cloud services in a uniform manner using our abstract model and has an initial low level view concerning metrics, composed of information provided by cloud APIs and monitoring tools. These two views on metrics are mapped by our elasticity control runtime, aggregating low-level metrics for computing higher level ones. For instance,

availability at service level would be computed from availability at each service part and the cost is aggregated from static information from the cloud provider on cost per I/O and VM cost, and the run-time service topology and loads.

4.1.4 Cloud Service Model Mapping on Existing Standards

Using the SOA reference architecture as the underlying architecture for the IBM CCRA [61], OpenGroup defines cloud services as being similar to SOA services in terms of governance, integration, interfacing with the consumer and construction. This is why the services (which are also referenced as cloud applications or artifacts [61]) are shown as being composed of service units. However, a distinction is made between cloud services which can be viewed as software, platform, infrastructure or business, and applications added by cloud service customers on the cloud using any of these services.

For this model we do not make a distinction between cloud services which are offered to the cloud consumer and the applications deployed on the cloud since both of them need to be elastically controlled and can be modeled considering the basic unit defining them, be it service unit, queue, Web service, etc. The cloud services are usually defined considering the aforementioned architectures and a description language (i.e., TOSCA). By specifying the application structure and the metrics he/she is interested in, the cloud service can be translated into our own model since many of the concepts used in our model (like service topology, service unit, etc.) were inspired from the existent architectures and languages. However, this mapping from description languages to our abstract model is out of the scope of this thesis.

4.2 Case Study Application

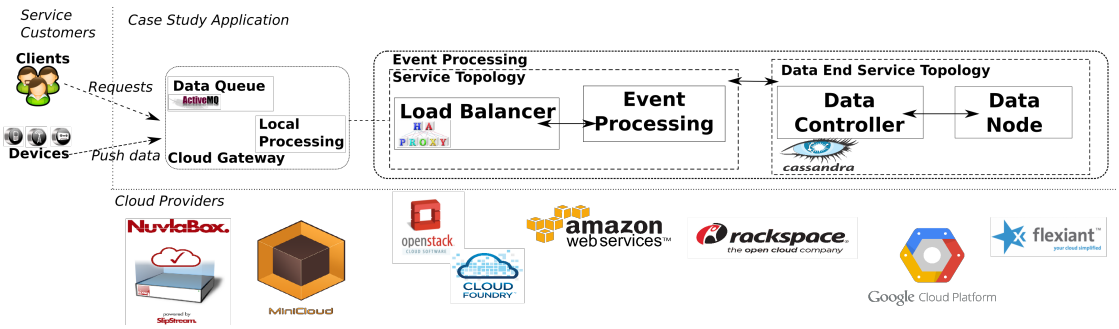


Figure 4.4: Case study application

The case study application is a Machine-to-Machine Data as a Service (M2M DaaS) application[2], developed as a pilot application for research in Distributed Systems Group, TU Wien. The application, shown in Figure 4.4, is supports both cloud and IoT case studies, simulating through the `Local Processing Service Topology`, IoT

[2] `https://github.com/tuwiendsg/DaaSM2M`

gateways which get the data from IoT sensors through an ActiveMQ[3] queue. The `Local Processing Service Topology` contains a lightweight Java process (i.e., `Local Processing Service Unit`) which evaluates the buffered data, and considering its speed, decides whether it should send it forward to the cloud.

In the cloud, we have an `Event Processing Service Topology`, which consists of a HAProxy[4] balancer (i.e., `Load Balancer Service Unit`), and web service (i.e., `Event Processing Service Unit`) which processes information coming from the `Local Processing Service Topology`. This data is processed and stored in a Cassandra NoSQL database cluster[5] (i.e., Data End Service Topology). The `Event Processing Service Unit` automatically creates new column families, depending on the structure of the data sent from the `Local Processing Service Unit`.

This application can be used both partially and as a whole, by simply replacing the other part of the application with a workload generator. For instance, on application's GitHub repository there are three configurations available: (i) the entire application, (ii) the Event Processing Service Topology and the Data End Service Topology, and (iii) the Data End Service Topology. Of course, each of these configurations also have appropriate workload generators, the first one replacing the IoT devices or customers, the second one replacing cloud customers and Local Processing Service Topology, and the third one replacing the Event Processing Service Topology. Throughout the thesis this application is used either as a whole, or in one of the three configurations, in order to better emphasize the presented scientific contributions.

The application's behavior can be changed during runtime as follows:

- one can add/remove `Local Processing Service Unit` artifacts (with the necessary resources associated)
- change the balancing type on HAProxy
- add/remove `Event Processing Service Unit` artifacts (with the necessary resources associated)
- add/remove Cassandra nodes (i.e., `Data End Service Unit`)
- re-balance Cassandra cluster (i.e., `Data End Service Topology`)

All elasticity capabilities above are composed of several primitive actions (e.g., for removing a Cassandra node, one has to first decommission data to other nodes, then remove network interfaces, virtual machines, or disks), and are modeled according to the model presented in Section 4.1. Further details concerning this application (e.g., elasticity requirements) will be discussed in the following chapters.

[3] `http://activemq.apache.org/`
[4] `http://www.haproxy.org/`
[5] `http://cassandra.apache.org/`

CHAPTER 5

Elasticity Requirements Language

This chapter presents the SYBL language, through which cloud stakeholders can specify multi-dimensional elasticity requirements (i.e., concerning cost, quality, and resources) at various levels of abstraction (i.e., code region, service unit, service topology, cloud service).

5.1 Overview

Although elasticity plays an important role in nowadays cloud computing infrastructures, it is typically regarded only from the resources elasticity point of view [19, 85, 109]. For instance, Amazon[1] provides the autoscale service, which allows the definition of policies for automatically scaling the service. ElasticHosts [2] and CloudSigma[3] are two of the providers which have, as selling point for their infrastructures, assuring elasticity. However, elasticity is a multi-dimensional view, such as one can scale up/down the quality or the cost of the services, rather than just the resources, as shown in recent studies [35]. Unfortunately, little focus has been devoted to defining elasticity and controlling it from a multi-perspective view and to allowing customers to specify complex and changing elasticity requirements throughout their service's execution on a cloud infrastructure.

In our work, we consider elasticity a multi-dimensional issue, defined by the relation between elastic properties classified into three main dimensions namely cost, quality and resource. In this view, the user should be able to specify constraints and relations for the aforementioned dimensions. To this end, our approach is to develop programming elasticity directives. Programming directives are well known, simple-to-use, and efficient means of controlling work distribution and communication at runtime. We believe that similar directives but for controlling elasticity can also hide the complexity of writing and executing elasticity strategies from the user.

[1] http://aws.amazon.com/autoscaling/
[2] https://www.cloudsigma.com
[3] https://www.cloudsigma.com

In this chapter, we contribute the detailed design and implementation of SYBL (Simple Yet Beautiful Language) – a novel language for controlling elasticity in cloud services – and its runtime system for controlling elasticity in cloud services. SYBL can be used for three specification levels: cloud service level, service topology level, service unit level and programming level. The elasticity specification at service level gives a high level description of user's preferences regarding the service to be deployed on the cloud infrastructure. With a higher level of granularity, service unit level elasticity requirements refer to the service units constituting the service, while programming level elasticity requirements specification deals with code-level elasticity requirements description. The service unit level elasticity requirements have been tackled by existing research work so far [19, 47, 91], but without a clear focus on cloud service elasticity. To the author's knowledge, programming level elasticity requirements specification has not been approached before, mostly due to the complexity of elasticity in cloud computing.

Service developers, software providers, IaaS or PaaS end-users and cloud providers can use SYBL. This facilitates the control of elasticity of a multitude of services in which SYBL can be used. For example, a cloud customer can try to achieve trade-offs on cost and quality, or just try to minimize the price that customers have to pay. On the other side, SYBL can be used by cloud providers to specify generic elasticity strategies for the cloud services hosted on their infrastructures. Therefore, SYBL enables many types of users to specify the elasticity behavior of a service without having to use complex cloud APIs or monitoring tools.

5.2 Elasticity Requirements

Elasticity is usually referred-to just in terms of resources, without considering implications from the point of view of other properties a service may have [19, 85, 91]. When talking about elasticity we refer to the capacity of a service to change or to be changed according to the context in which it resides. Elasticity targets not just resources and their capacity to scale, but also their relations with the different types of costs and quality, and the capacity of an service to oscillate between different states (each state being described by resources, cost, quality and their attributes) for obtaining best quality at best price possible.

For describing what types of elasticity control could be required and the complexity of elasticity requirements, we examine elasticity requirements from different types of clients. Let us consider a generic service model of which the structure is given in Figure 5.1. The service has different service units, including an event processing service unit, and a data end service topology, being a part of the application described in Section 4.2. Each service unit consists of multiple processes, which at runtime may need to scale depending on user requirements that are parameterized by the number of users, the difficulty of computations and the type of processes. Each process of the service unit has certain elasticity requirements at deployment. Depending on the user-defined specifications, the processes can be scaled individually either by creating more process instances in the same or different virtual machines, or by allocating more or less computing resources.

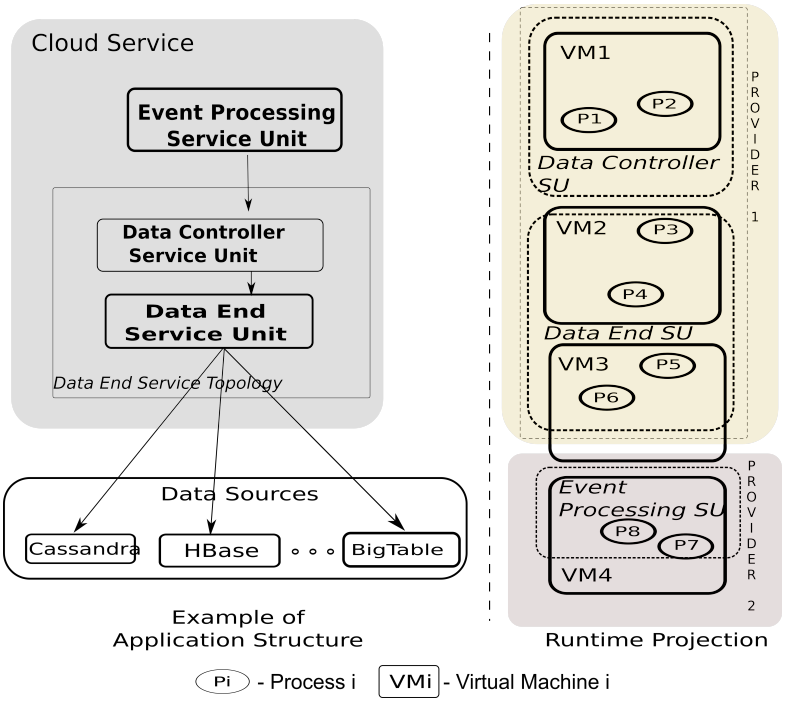


Figure 5.1: Illustrative service structure and its deployment

The elasticity requirements are demands formulated by users, containing conditions that are necessary for the service to be elastic. Elasticity can be a subjective notion and depends on the type of users and the granularity at which he/she wants to specify the service's elasticity. On one hand, the purpose of elasticity requirements varies from controlling costs to achieving higher quality or even specifying demands on the relation between cost, resources and quality, for example:

- *Cost-related elasticity requirements*: A cloud customer may specify that when the total cost is higher than 80 Euro, there should be a scale-in action for keeping costs in acceptable limits.

- *Quality-related elasticity requirements*: A cloud user may need to monitor different quality parameters, which should be in acceptable limits. For instance, a software provider can specify constraints on the response time depending on the number of users currently accessing the provided software. A developer could specify that the result from a data analytics algorithm must reach a certain data accuracy under a cost constraint without caring how many resources should be used for executing the code of that algorithm.

- *Elasticity requirements on the relation between cost and quality*: A cloud provider could specify its pricing schema or price computation policies, for example that when availability is higher than 99% the cost should increase by 10%.

On the other hand, the user needs different granularities at which he/she can specify elasticity requirements, therefore, having specifications enforced at different levels, as opposed to usual approaches in resource allocation and reallocation such as to control resources only at the whole service or service unit level [19, 56, 91]. To this end, elasticity control should be supported at the following levels:

- *Service level elasticity requirements*: elasticity requirements can be applied on the overall availability of the whole service, aggregating data about different service units and the communication between them. The service cost refers to the full cost of operating it: cost of service units' usage, communication between service units and storage.

- *Service unit level elasticity requirements*: the user could specify different requirements based on the service unit type, i.e. the nature of requirements for the computation engine that are different from the requirements for the front-end service unit. The user can control the cost associated with a service unit, which means aggregating the cost of processes, storage and communication associated to the service unit but residing on different virtual machines.

- *Programming level elasticity requirements*: the user could need to specify that for some specific code the CPU size and memory should be really high, when the cost is high. In this case, when specifying requirements about the cost, the developer refers to the cost for running the specific portion of code.

To support the above-mentioned requirements, we characterize elasticity properties into a "resources-cost-quality" representation in Figure 5.2 where each axis contains sub-dimensions of cloud service requirements specification. The user should be given the opportunity of specifying the service's behavior, most specifically in what direction it should automatically scale in different cases in the space defined by the three axes (resource, cost and quality). Many properties from each of the elasticity dimensions are strongly interdependent, an elasticity property belonging to one axis being a multi-dimensional function of properties belonging to the other two axes.

We, therefore, need a novel approach towards elasticity control specifications. Our approach is to use programming directives that can be used to monitor and specify different elasticity constraints and strategies. Programming directives for controlling elasticity should be easy to use. However, they should be generic and extensible.

5.3 SYBL Syntax and Semantics

The initial idea of using elasticity directives for cloud computing is first described in [34], targeting programming level directives in elastic computing. The directive-based elasticity specification can substantially reduce the overhead by delegating the control to underlying elasticity middleware. SYBL is designed for that purpose in order to meet requirements mentioned in Section 5.2. Before describing in detail how we design and implement

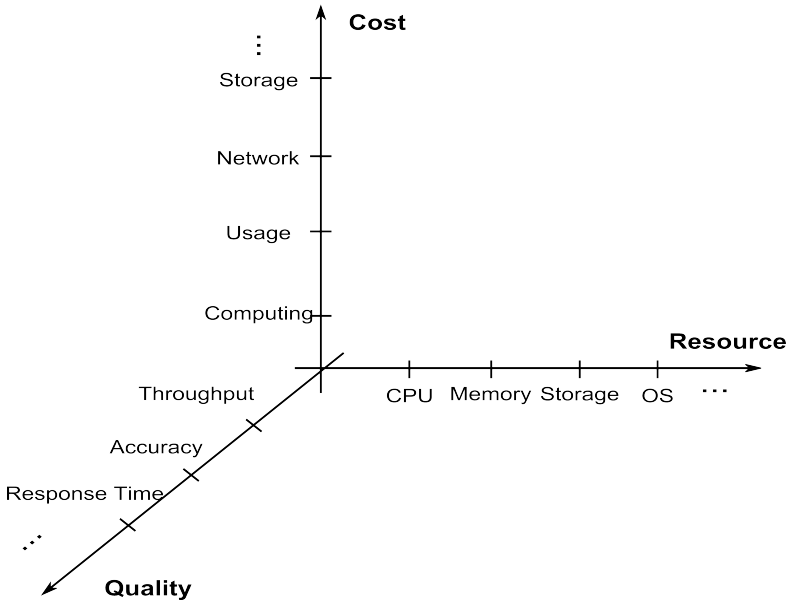


Figure 5.2: Common dimensions for service elasticity

directives for elasticity control, we will overview the concepts of elasticity programming directives in this section.

SYBL enables elastic specifications at different granularities, depending on the user's demands and perspective: service, service unit and programming (see Figure 5.1). As opposed to usual mechanisms of reallocations, where rules (mostly SLAs) are specified at a per-service level, SYBL provides greater elasticity granularity. At the highest level, the service level, global characteristics can be described. At the service unit level we can express a lower level description for black box service units or elasticity requirements focusing at a level lower than service but higher than code, while the programming level enables the user to specify elasticity requirements at code level. One may argue on the need of service level specifications, when the user can specify broadly all the requirements at programming and service unit level. This service level meets cloud users who may want to deploy their service as a black-box, or cloud providers who receive the service as a black box without having any permission to access it. By providing these finer-grained specification levels, we enable the user to decide where elasticity specifications could or should be placed, considering service's elasticity requirements.

The SYBL language, being a directive-based one, is easy to understand and use, while aiming of high expressiveness. The elasticity specification described in the previous section would be a strategy "`averageCost` > 30 Euro: scaleIn". This would simplify the complexity of using the cloud API and multiple monitoring tools calls to implement elasticity control as well as enable the multi-dimensional elasticity control, which are not well supported currently. Using programming level elasticity directives, the user can focus on defining the strategy and on the appropriate measures he/she should take.

SYBL empowers the user to design elasticity specifications that reference other

specifications. After defining a complex constraint, the user can simply specify strategies about its violation or fulfillment, without having to re-describe it as a condition for the new strategies. It also enables a hierarchical description of elasticity specifications, for cases of overlapping constraints or strategies or unclear elasticity descriptions.

5.3.1 Language Constructs

Predefined Functions

SYBL includes several predefined functions which regard two different kinds of information: information on the current environment and on the elasticity specifications. The environment comprises different types of static and dynamic cloud information. When we refer to the environment, we have to consider the level at which each of these predefined functions or variables appear. As described before, the information differs in the different levels of elasticity requirements, and therefore the environment that comprises all the information at a given moment in time will also differ based on these levels. Accessing information on the current environment enables the user to be aware of capabilities of the underlying cloud computing infrastructure and the current service behavior.

Function	Description
GetEnv	Current cloud infrastructure environment
Violated	Checks whether the constraint sent as parameter is violated
Enabled	Checks whether an elasticity specification is enabled or not
Priority	Returns the priority of an elasticity specification

Table 5.1: Example of predefined functions

SYBL contains several predefined functions like *GetEnv* which can be used to obtain the current environment (taking also into account the level from which it is called), $Violated/Fulfilled$ returns true or false, depending on whether or not the constraint received as a parameter is fulfilled, the *Priority* function associates to each elasticity specification a priority (see Table 5.1). The environment variables refer to general information like the currently considered compute bid, the cost spent, etc. (see Table 5.2). The environment variables depend on the level at which they appear in elasticity specifications. The environment variables have at their source more complex function calls which are used frequently. For example, `optimal_cloud_provider` variable hides a call to `GetEnv().findOptimalCloudProvider()`.

Equations 5.1 and 5.2 formally describe the set of variable and functions predefined in SYBL, which will be later used for the description of monitoring, constraints, and strategies.

$$DefFunctions := \{GetEnv, Balance, Violated, Enabled, Priority\} \tag{5.1}$$

$$EnvVariables := \{compute\_bid,\ total\_cost, optimal\_cloud\_provider\} \tag{5.2}$$

Environment variable	Description
optimal_cloud_provider	The cloud provider that the decision service units finds to be best suited
compute_bid	The current bid for the current cloud provider
total_cost	The cost - depends on the level at which variables are being referenced

Table 5.2: Examples of predefined environment variables

Monitoring Directives

The monitoring directives start with the `MONITORING` keyword and assign to a new variable some types of cloud information to be monitored (see Equation 5.3). The monitoring variable is assigned existing information or formulas constructed for combining several types of cloud information.

$$M_i := \texttt{MONITORING}\ varName = x_j \mid \texttt{MONITORING}\ varName = formula(x_1 ... x_n) \\ where x_j \in c, c \in\ ServiceDescriptionInfo \tag{5.3}$$

Constraints directives

A constraint describes the limits in which the current service's description can oscillate. As shown in Equation 5.4 a constraint directive starts with the keyword `CONSTRAINT`, and uses mathematical signs ($<, >, >=, <=, !=, ==$) for reflecting which values are admissible. Constraints can be established on a simple type of cloud information or on a complex type of cloud information determined by formulas ($formula_i$ or $formula_j$).

$$C_i := \texttt{CONSTRAINT}\ p\ \in\ formula_i(x)\ rel\ formula_j(y) \\ where\ x, y\ \in\ ServiceDescriptionInfo rel\ \in\ \{\leq, \geq, \neq, =\} \tag{5.4}$$

Strategies Directives

A strategy describes a recipe to be followed in case the triggering condition becomes true. A strategy starts with the keyword `STRATEGY` and usually has the form Condition:Action or `WAIT` Condition (see Equation 5.5). The first pattern triggers the execution control action (e.g., deploy, migrate, delete, scale) specified in case the condition is true, while the second pattern waits for the condition to be true. The condition can also refer to fulfillment or violation of constraints, and actions to be taken in those cases.

$$S_i := \texttt{STRATEGY CASE}\ [Condition : Action] \mid \texttt{WAIT}\ Condition \mid \texttt{STOP} \mid \texttt{RESUME} \mid \\ \texttt{EXECUTE}\ strategyName\ parameter_{1...n} \\ whereCondition : DefFunctions \rightarrow \{true, false\} \tag{5.5}$$

5.3.2 BNF Form of SYBL

SYBL facilitates the description of elasticity requirements at different levels, depending on the service provider's knowledge on the cloud service and on his/her perspective: cloud service, service topology, service unit, elasticity relationship, and programming/code region level. SYBL is implemented as directives in different languages, enabling easy description of the requirements, and delegating the actual difficult part of controlling the cloud service to the SYBL runtime (rSYBL), which is the controller of the cloud service.

Listing 9.1 shows in BNF the constructs of the SYBL language. The *monitoring* directives start with the `MONITORING` keyword and specify new variables to be monitored. A *constraint* defines elasticity requirements for the cloud service state, defining the limits of cloud service behavior. A *strategy* specifies requirements on the elasticity behavior of the service. It specifies both control strategies to be enforced under specific conditions, and `WAIT`, `STOP` or `RESUME` actions for the controller, which can be paused/stopped/resumed when specified conditions hold. Therefore, with these two constructs at the center of the SYBL language, *constraints* and *strategies*, depending on the service provider/developer knowledge about the service, we enable various elasticity state and behavior specification mechanisms, the controllers interpreting the language, detailed in Section 6.3, being in charge with determining the specific control mechanisms which enable such service states or behaviors.

Listing 5.1: SYBL in Backus Naur Form (BNF)

```
Constraint := constraintName : CONSTRAINT ComplexCondition
Monitoring := monitoringName : MONITORING varName=MetricFormula
Strategy := strategyName : STRATEGY CASE ComplexCondition :
                action(parameterList) |
          strategyName : STRATEGY WAIT ComplexCondition |
          strategyName : STRATEGY STOP ComplexCondition |
          strategyName : STRATEGY RESUME ComplexCondition
MetricFormula := metric | number | metricFormula MathOperator
        metric | metricFormula MathOperator number
ComplexCondition := Condition | ComplexCondition
    BitwiseOperator
        Condition|(ComplexCondition BitwiseOperator Condition)
Condition := metric RelationOperator number| number
        RelationOperator metric | Violated(name)|Fulfilled(name
            )
MathOperator := + | - | * | /
BitwiseOperator := OR | AND | XOR | NOT
RelationOperator := <|>|>=|<=|==|!=
```

SYBL hides the complexity of enforcing a variety of complex calls, to different APIs (e.g., cloud provider APIs, or bash configurations) with the help of elasticity capabilities defined in the model in Section 4.1. It facilitates the service provider/developer

to focus more on the elasticity requirements that would help his/her application to behave as desired. For referring to the current used infrastructure or platforms, it offers several predefined functions and environment variables with pre-defined semantics. The environment comprises different types of static and dynamic cloud information, its capabilities (e.g., whether or not it can modify the service during runtime and in what extent), as well as service-related information. When referring to the environment (e.g., through the predefined function *GetEnv*), the stakeholder needs to consider the level at which functions or variables appear, since information and extent of control varies with the level at which the SYBL elasticity requirement is specified. For instance, at service topology level the service provider/developer would get environment information regarding his/her service topology, which might be running in a different region than the rest of the cloud service.

5.3.3 Elasticizing Cloud Services with SYBL Elasticity Requirements

The SYBL language is not strictly linked to any specific implementation language (e.g., they can be seen as Java annotations, C# annotations, or Python decorators). Moreover, the SYBL elasticity requirements can be injected into any cloud service description language (e.g., TOSCA) or can be specified separately through XML description. Current language interpretation mechanism is implemented in Java, and supports TOSCA-injected, XML-based, or Java annotation-based elasticity requirements specification.

For example, Listing 5.2 shows a constraint specified for the service topology with ID `WebService Topology`. The elasticity requirement sets the preferred response time below 450 ms. We define this elasticity requirement as a subtype of Java Annotation, triggered at runtime when the annotated method is executed, and caught and interpreted using AspectJ.

Listing 5.2: Example of elasticity requirements as Java Annotations

```
@SYBL_ServiceTopologyDirective(annotatedEntityID=
                    "WebServiceTopology", constraints=
                    "Co3:CONSTRAINT␣responseTime␣<␣450␣ms;")
```

Listing 5.3 shows a strategy for the service topology with ID `DataEnd Topology`. The elasticity requirement is a conditional strategy, which enforces the action `scalein` for the service topology when both responseTime and the average throughput are above predefined values.

Listing 5.3: Example of elasticity requirements in XML

```
<SYBLSpecification id="DataEndTopology" type="ServiceTopology">
   <Strategy Id="St1">
     <Condition>
        <BinaryRestriction Type="smallerThan">
           <LeftHandSide>
             <Metric>throughputAverage</Metric>
```

```
            </LeftHandSide>
                        <RightHandSide>
                                <Number>300</Number>
                        </RightHandSide>
                </BinaryRestriction>
                <BinaryRestriction Type="smallerThan">
                        <LeftHandSide>
                           <Metric>responseTime</Metric>
                         </LeftHandSide>
                                <RightHandSide>
                                        <Number>360</Number>
                                </RightHandSide>
             </BinaryRestriction>
          </Condition>
   <ToEnforce ActionName="scalein" />
 </Strategy>
</SYBLSpecification>
```

The constraint shown in Listing 7.1 specifies that the cost for the `PilotCloudService` should be below 100$. The SYBL elasticity requirements can be easily integrated within TOSCA policies, and interpreted by the elasticity controller.

Listing 5.4: Example of elasticity requirements as TOSCA Policies

```
<tosca:ServiceTemplate name="PilotCloudService">
  <tosca:Policy name="St1" policyType="SYBLStrategy">
    St1:STRATEGY minimize(Cost) WHEN high(overallQuality)
  </tosca:Policy>...
```

5.3.4 SYBL Runtime

The SYBL runtime takes elasticity specifications and carries out elasticity control at runtime. In the SYBL runtime, elasticity requirements expressed through SYBL will be interpreted, processed by the *Control Service* and then enforced by the use of cloud APIs. The *Control Service* is the central part of the runtime system, being the service unit which handles the actual coordination between the specified state of the service from user's perspective, and the current service elasticity state.

The deployment of SYBL runtime system from Figure 5.3 mainly concerns the SYBL programming directives, which are more difficult to be enforced since they need to be caught with higher precision, but also applies to service unit and service level directives. The *SYBL Local Interpreter* is instantiated within each process containing SYBL directives. The *SYBL Local Interpreter* catches the SYBL directives, interprets them and forwards the requests to the *Local Service*. The latter is a part of the *Control Service*, is deployed and resides on the VM and enforces the different elasticity requirements coming from

processes of the same service. The main *Control Service* communicates with *Local Services* and correlates received information for enabling the enforcement of requirements at service unit level and service level, even when the artifacts or processes do not reside on the same VM. The *Control Service* also communicates with cloud APIs and monitoring tools for having an up-to-date knowledge about the service elasticity state.

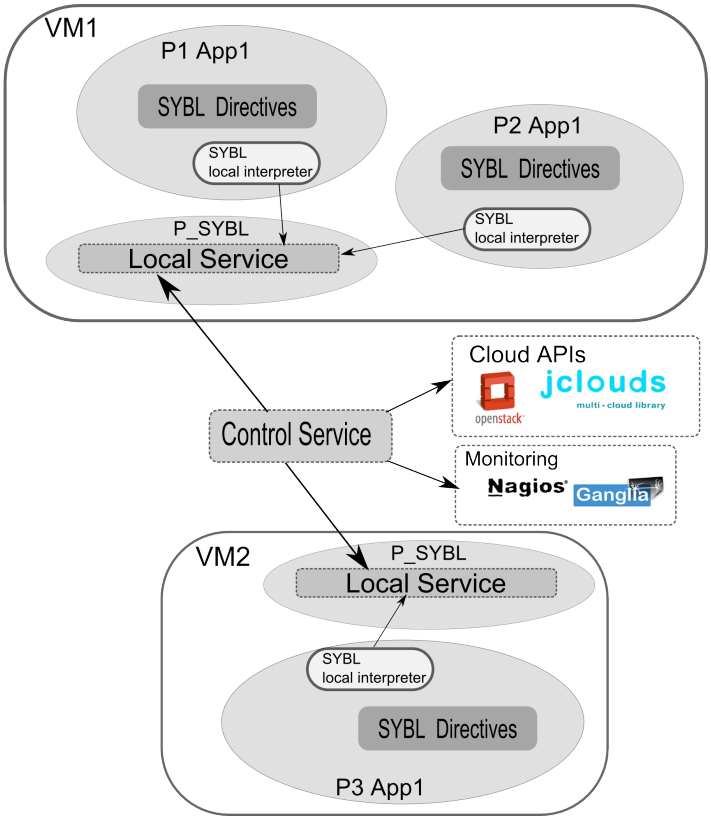


Figure 5.3: SYBL based control at runtime

The *Control Service* uses monitoring tools and cloud APIs for providing the necessary functionality. The multitude of cloud APIs and monitoring tools that elasticity requirements depend on can seem overwhelming. The need for more than one tool with which the SYBL runtime interacts is obvious, since a single tool could not provide a complete description of the cloud infrastructure capabilities and generic mechanisms of monitoring and interacting with the cloud infrastructure. For instance, Nagios[4] and Ganglia[5] are many times used together due to the fact that they compensate each-other in the information provided and the mechanism of collecting data.

These tools are in a continuous change and this is why the language should not be bound to a specific cloud API. However, the *Control Service* should be able to interface

[4] `https://www.nagios.org`
[5] `http://ganglia.sourceforge.net/`

to the tools specific to different cloud providers (e.g. via plug-in mechanisms). SYBL is designed to be extensible: that is, to have the capacity of enveloping new concepts without much difficulty. SYBL users can easily add metrics and concepts they need to focus on, the extensibility property being one of the strong points of SYBL language.

5.3.5 Examples of Elasticity Control

The following SYBL examples do not refer to a specific language implementation, the intention being simply to show that short and simple SYBL elasticity requirements hide the actual complex implementation and enforcement layers.

Listing 5.5 shows how a cloud provider can actualize the price perceived for the current service depending on service's availability so far. The strategy $Str1$ specifies the price for the case in which the availability is greater than 98%. The strategy $Str2$ overrides the previous specification, stating that an availability larger than 99% should set the price at 300 Euro.

Listing 5.5: SYBL elasticity requirements - cloud provider

```
#SYBL.ServiceLevel
Str1: STRATEGY CASE availability>98% setPrice(10)
Str2: STRATEGY CASE availability>99% setPrice(30)
Priority(Str1)<Priority(Str2)
```

Listing 5.6 shows possible elasticity requirements from the service developer side. Constraint $ServiceUnit2.Cons5$ specifies that the CPU usage in the computation engine should be less than 80% for avoiding performance degradation due to high CPU load. Constraints $Cons3$ and $Cons4$ overlap in the sense that both refer to the costs, but at different levels: $Cons4$ specifies that the cost of hosting the `Data End Service Unit` should be less than 60 Euro, while $Cons3$ refers to the cost of hosting the entire service. The programming level elasticity requirement encompasses the sequence of code, which for this case can be a data analysis algorithm, setting constraints about data accuracy and costs, without having specific resource-related requirements.

Listing 5.6: SYBL elasticity requirements - developer

```
#SYBL.ServiceLevel
Mon1: MONITORING rt = Quality.responseTime
Cons1: CONSTRAINT rt < 2 ms. when nbOfUsers < 1000
Cons2: CONSTRAINT rt < 4 ms. when nbOfUsers < 10000
Cons3: CONSTRAINT totalCost < 80 Euro
Str1: STRATEGY CASE Violated(Cons1) OR Violated(Cons2):
    ScaleOut
Priority(Cons1)=3, Priority(Cons2)=5
#SYBL.ServiceUnitLevel
ServiceUnitID = ServiceUnit3; Name= DataEndServiceUnit
Cons4: CONSTRAINT totalCost < 60 Euro
```

```
#SYBL.ServiceTopologyLevel
ServiceUnitID = ServiceUnit2; Name= DataEndServiceTopology
Cons5: CONSTRAINT cpuUsage < 80%
#SYBL.ProgrammingLevel
Cons6: CONSTRAINT dataAccuracy>90% AND cost<400
```

These examples show the ease of specifying elasticity requirements with SYBL and the power that lays underneath the simple description. All these elasticity requirements are enforced with no effort from the user side, and transformed into calls to different APIs and tools that help at enforcing these elasticity requirements.

For cases where users (software providers and cloud providers) have a black box service/service unit for which they need to specify elasticity requirements, elasticity specifications can be annotated inside the XML-based descriptions of the service/service unit (e.g. OVF). Listing 5.7 shows how a specific elasticity specification can be integrated into the already existing sections of the OVF format. The resource ranges can be integrated into the already existing OVF structure, e.g. the specification bellow states that the minimum amount for memory (OFV resource type with value 4) should be greater than 384 MB. The elasticity requirements for quality and cost are added as an XML structure in the additional section of OVF.

Listing 5.7: Example of elasticity constraints in OVF

```
<VirtualHardwareSection><Item ovf:bound="min">
        <rasd:InstanceID>0</rasd:InstanceID>
        <rasd:Reservation>384</rasd:Reservation>
        <rasd:ResourceType>4</rasd:ResourceType>
</Item> </VirtualHardwareSection>
```

5.4 Experiments

We have developed a prototype of SYBL[6] containing a partial implementation of its runtime system. We currently support the SYBL elasticity specifications for processes residing inside the same VM, the implementation of the *Control Service* (Figure 5.3) being in our focus as future work. We support SYBL elasticity specifications as Java annotations processed at runtime by AspectJ or as SYBL enriched XML descriptions, which are interpreted and then enforced through the *Local Service*. We tested the current prototype on our local cloud running OpenStack[7], . In the current implementation, Ganglia[8] provides information on computing resources allocated to virtual machines and their usage, the number of packets sent and received by each virtual machine, etc. JClouds [9] is used mainly for scaling the current instances and for controlling them.

[6]Detailed prototype implementation and further experiments can be found at http://dsg.tuwien.ac.at/prototypes/SYBL/index.html.

[7]https://www.openstack.org/

[9]https://jclouds.apache.org/

5.4.1 Experimental Application

The structure of our experimental application is shown in Figure 5.4 and described in Section 5.2. In our experiments we focus on illustrating the feasibility of SYBL specifications. The application is composed of the `Data End Service Topology` presented in Chapter 8.2, for which we use for generating workload the YCSB benchmark [22].

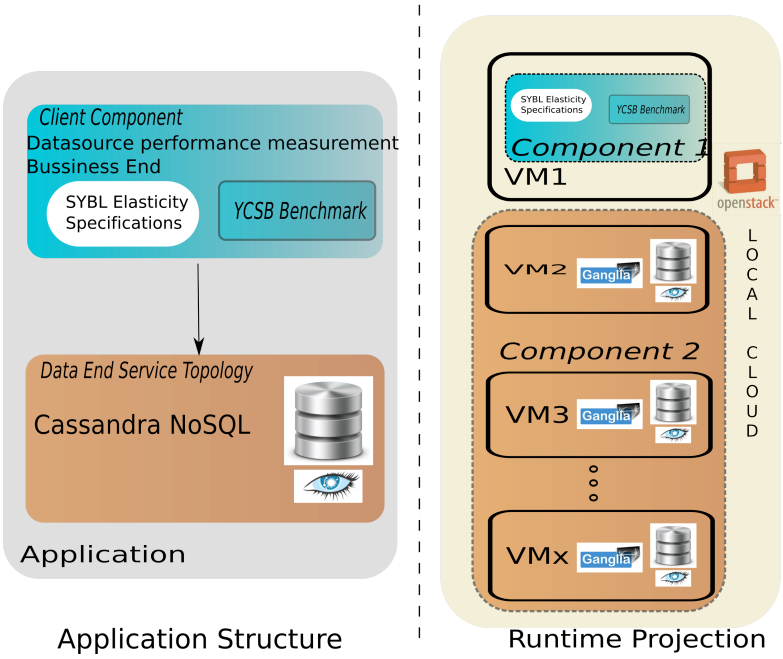


Figure 5.4: Application structure used for experiment

For this chapter's experiments we focus on the Data End Service Topology described in Section 4.2 and its ability to be dynamically controlled by the SYBL user. As data engine we use Cassandra - a NoSQL distributed database system, while at the business side we use the YCSB d benchmark for simulating different types of workload (configuration (iii) from Section 4.2). The application is mainly composed of a virtual machine hosting the business end of the application and a cluster of virtual machines hosting Cassandra nodes. The workloads used represent a combination of read, write, scan and update operations, actually characterizing real-life applications. For example, a read-oriented workload can be related to an application focused on photo-tagging, where the largest part of the operations is reading tags. The user of this service may have the following elasticity requirements:

- *Service level elasticity requirement*: the user may need for the overall cost of his service being hosted for this experiment to be less than 20 Euro
- *Service unit level elasticity requirement*: the user could specify strategies for the

case in which the average number of IOs is too small, this way generating one more business service unit and therefore more workload

- *Programming level elasticity requirement*: the user may want for his service data-end to scale dynamically when running the workload, for keeping the CPU usage in admissible values.

5.4.2 SYBL Annotations-based Elasticity Requirements

The SYBL annotation (see Listing 5.8) states at service unit level that in case the average number of IOs is less than 50 service should scale out (strategy $St1$) for introducing a higher workload. Strategies $St2$ and $St3$ of the service unit level annotation enforce the two constraints specified, for keeping resource utilization in acceptable ranges.

Listing 5.8: Service unit-level SYBL annotation

```
@SYBL_ServiceUnitContext(constraints=
"Co1:CONSTRAINT␣memory.usage<80%␣AND␣cpu.usage<80%;
Co2:CONSTRAINT␣memory.usage>20%␣AND␣cpu.usage>20%"
,strategies=
"St1:STRATEGY␣CASE␣IO.averageNb<50:ScaleOut;
St2:STRATEGY␣CASE␣Violated(Co1):␣ScaleOut;
St3:STRATEGY␣CASE␣Violated(Co2):␣ScaleIn")
```

The programming level SYBL specification from Listing 5.9 annotates a method executing workload that generates database operations. The user specifies an elasticity requirement that the data source should automatically scale and keep the CPU usage on the data engine at predefined levels. From the three specified constraints just two are always enabled. This is due to the higher priority of constraint $Co3$ next to the constraint $Co1$. The strategies refer to the constraints giving scaling advises with respect to the data source for the two mentioned cases. As a result of producing the workload referred by this method, the database will scale dynamically as we will see in the next subsection.

Listing 5.9: Programming-level SYBL annotation

```
@SYBL_ProgrammingContext(type=AnnotType.DURING,
 constraints="Co1:CONSTRAINT␣cpuUsageData␣<␣65;
␣␣Co2:CONSTRAINT␣cpuUsageData␣>␣30;
Co3:CONSTRAINT␣cpuUsageData␣<␣85␣WHEN␣cost␣>␣70",
 monitoring ="Mo1:MONITORING␣cost␣=␣cost.instant;
␣␣Mo2:MONITORING␣dataThroughput␣=␣throughput.datasource;
␣␣Mo3:MONITORING␣cpuAllocatedData␣=␣cpu.size.datasource;
␣␣Mo4:MONITORING␣cpuUsage␣=␣cpu.usage;
␣␣Mo5:MONITORING␣cpuUsageData␣=␣cpu.usage.datasource",
 strategies=
 "St1:STRATEGY␣CASE␣Violated(Co2):␣scaleInDataSource;
␣␣St2:STRATEGY␣CASE␣Enabled(Co1)␣AND␣Violated(Co1):
```

```
                                    scaleOutDataSource;
  St3:STRATEGY CASE Enabled(Co3) AND Violated(Co3):
                                    scaleOutDataSource;",
 priorities="Priority(Co3) > Priority(Co1)")
```

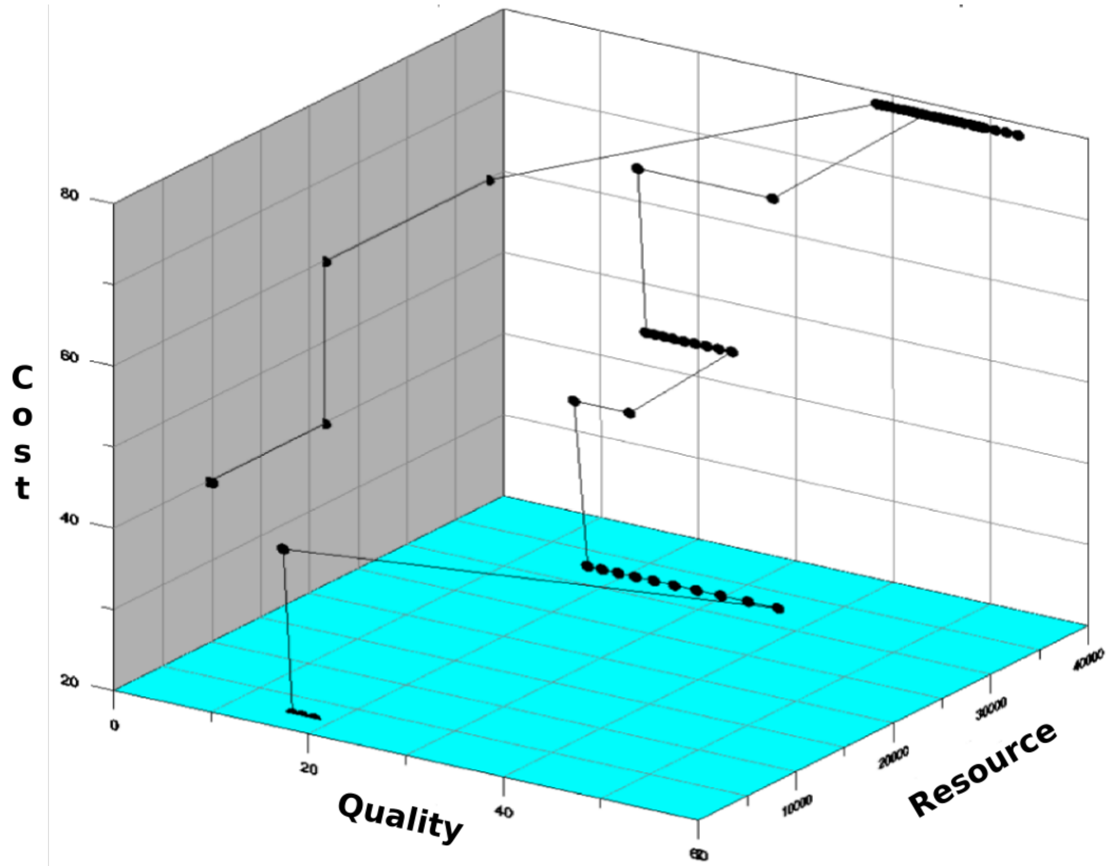


Figure 5.5: Evolution of Data End Service Topology in elasticity space

5.4.3 Results

Based on elasticity directives specified in Listing 5.9, the elasticity runtime behavior of the data engine is scaled on multiple instances hosting Cassandra nodes, considering the CPU usage and cost metrics. Figure 5.5 shows the elasticity evolution in terms of cost, quality and resources, each three-dimensional point being a three-dimensional state of the data source part of the service. The quality in this case refers to the throughput, the resources refer to the allocated CPU size while the cost is estimated based on the resources allocated. The chart shows how based on the SYBL elasticity annotations the service evolves in the elasticity space, adjusting the resources, cost and quality to the user's elasticity requirements. Figure 5.6 gives a view on the evolution of the service on

the data engine in terms of number of instances and CPU used: with the increase of instance number, produced by the SYBL strategies, the CPU usage decreases.

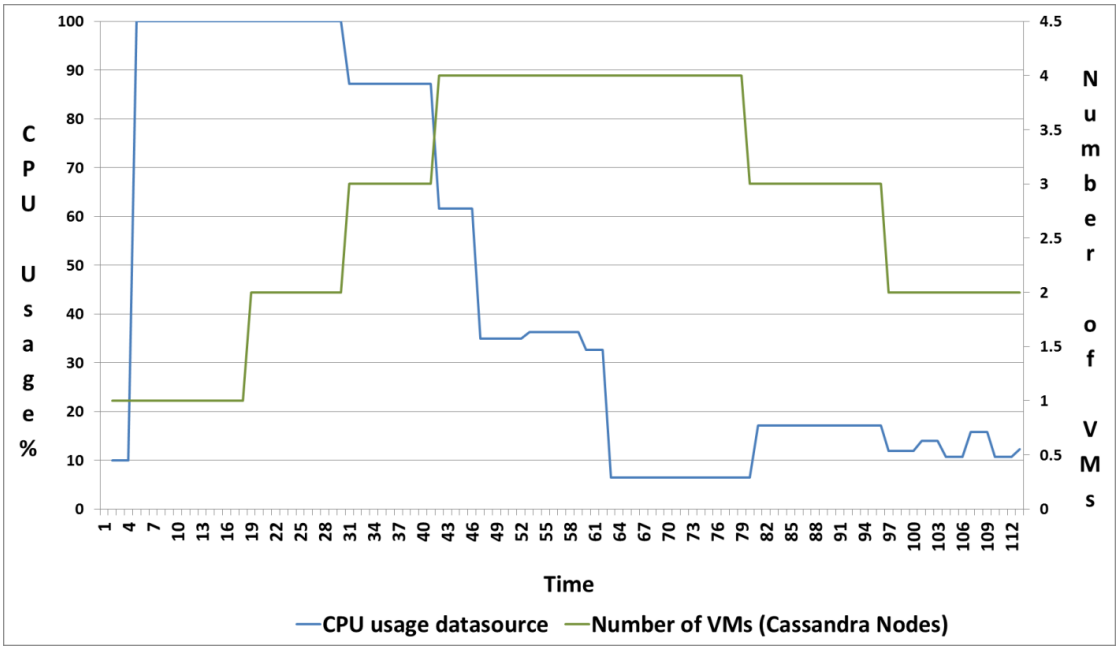


Figure 5.6: CPU usage correlation with the number of VMs used

This experiment reveals SYBL's strength, enabling users to describe the service elasticity requirements from the three perspectives: monitoring, strategies, and constraints. We have shown how SYBL and its runtime system can ensure through the business level elasticity specifications the elastic control of data sources, the user being able to specify his/her requirements at the desired granularity, and depending on the data that needs to be taken into consideration. The results show that the service elasticity has accorded with specified SYBL directives following accurately the desires of SYBL users.

CHAPTER 6

rSYBL : a Framework for multi-level Cloud Service Elasticity Control

This chapter presents the rSYBL framework, and mechanisms used by it, for generating action plans for controlling the elasticity of cloud services. The rSYBL control mechanisms are guided by elasticity requirements, specified by various stakeholders at multiple abstraction levels, using SYBL.

6.1 Overview

Cloud services can range from web applications, to workflows, and scientific applications of different types [39, 121, 127]. When experimenting and/or deploying them (ideally, automatically) on various cloud infrastructures, the cloud service provider/developer usually has high level, specific goals, e.g., testing reliability or achieving a specific level of performance with a minimum cost, at different levels of the service, e.g., for the entire data end or for specific parts of the data end. Current control frameworks mainly focus on single types of services and enable the provider/developer to only specify resource level SLA [19, 69]. Furthermore, they lack means to interact with the stakeholders for controlling the elasticity trade-offs, e.g., the service provider cannot change requirements during runtime [4].

As there are various types of stakeholders interested in cloud-hosted services (e.g., cloud service developers and cloud service providers), they have different preferences at the various abstraction levels. They have coarse or fine grained knowledge about their cloud services, with regard to various matters (e.g., the provider knows how much s/he is willing to pay for the entire service to be hosted on the cloud, while the developer knows quality indicators at different layers of the service). Therefore, there is a strong

need for mechanisms to specify multi-level elasticity requirements, customized for various parts of the cloud service. To address these requirements, we need to develop means for multi-level elasticity requirements specification targeting high level goals referring to not only resources but, more importantly, to quality and cost, following the multi-dimensional definition of elasticity [35]. Moreover, we need to manage both the static description of the cloud service, and its runtime behavior, which depends on the virtual infrastructure on which it runs.

We present our approach for multi-level elasticity control that generates action plans considering the evolution of the service at different levels of abstraction. To this end, we present our mature rSYBL framework, which is easily extendible, allows stakeholders to change their requirements during runtime, and supports multiple enforcement mechanisms (e.g., multiple clouds, and multiple software platforms), multiple monitoring tools, and planning mechanisms. We run experiments comparing rSYBL elasticity control on two clouds, one private based on OpenStack[1], and the Flexiant[2] public cloud infrastructure. We showcase an experimental evaluation on the importance of multi-level service control, and analyze the performance of rSYBL under two different cloud infrastructures (i.e., OpenStack and Flexiant).

6.2 Managing Elasticity Capabilities from Cloud Providers

The model above uses the *Infrastructure System Information* for enabling the elasticity controller to describe, understand and manage the runtime information and its relation with service units, service topologies and the entire cloud service. All elements that are part of the Infrastructure System Information can have associated elasticity capabilities, described as part of the *Elasticity Information*. Most cloud providers implement similar concepts describing the services they offer (e.g., Flexiant[2] offers servers, Amazon[3] offers instances, and Google[0] offers virtual machine, Amazon offers Elastic Load Balancing while Google offers Global Load Balancing, although both refer to distributing incoming requests across pools of VMs). Therefore, we use the concepts presented in the above model for the Infrastructure System Information, in order to describe the services used from the chosen cloud providers. Moreover, we can abstract possible elasticity capabilities for all resources belonging to the Infrastructure System Information, in order to be referred by other elements of the model or by the cloud service elasticity controller.

All the elements that are part of Infrastructure System Information have associated, during runtime, (i) the properties that are used by the respective cloud service parts (e.g., service unit, service topology, or the entire cloud service), and from the Elasticity Information part, (ii) the elasticity capabilities that are available for the cloud service controller to manage them, together with the mechanisms for triggering these control capabilities, and (iii) elasticity metrics that give the controller the necessary information

[1] `http://openstack.org/`
[2] `http://www.flexiant.net`
[3] `http://aws.amazon.com/ec2`

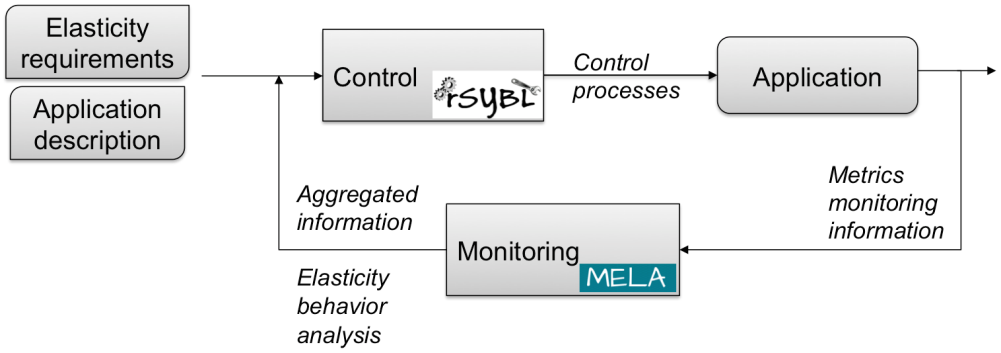


Figure 6.1: Feedback loop for controlling cloud service elasticity

in order to take control decisions. The elasticity capabilities of the service units, service topologies and cloud services are composed of a list with elasticity capabilities of different resources from Infrastructure System Information and are enforced by the cloud service elasticity controller.

6.3 Multi-level Elasticity Control

Based on the elasticity requirements specified at multiple levels of the service using the language presented in Chapter 5, the elasticity controller needs to leverage elasticity capabilities from multiple levels of the cloud service, from infrastructure to service topologies, and control the service for fulfilling the multi-level elasticity requirements. Generally, we are using a feedback loop (see Figure6.1), in which the cloud service is the plant, the controller is our controller, and the sensor is MELA [89], a tool for analyzing and monitoring cloud services.

6.3.1 Steps in multi-level elasticity control

Considering the model of the cloud service described through the runtime dependency graph presented in the Section 4.1, we enable elasticity control simultaneously for each of the described nodes, resulting in a multi-level elasticity control of the described cloud service, based on the flow shown in Figure 6.2. The service provider/developer describes his/her cloud service using TOSCA or other description standards. The initial deployment configuration is specified either by the automatic deployment tool used or by the service provider/developer if a manual deployment approach is chosen. The elasticity requirements are evaluated and conflicts that may appear among them are resolved. After that, an action plan is generated, consisting of elasticity capabilities that enable the fulfillment of specified elasticity requirements. The action plan is composed from elasticity capabilities that have associated a series of IaaS calls, configurations, or bash/scripts executions.

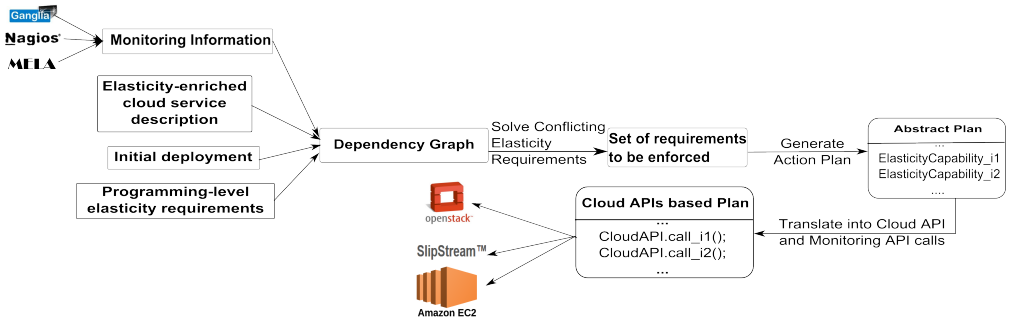


Figure 6.2: Elasticity control - from requirements to enforced plans

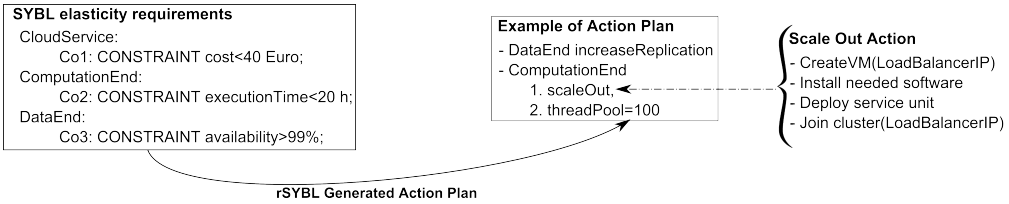


Figure 6.3: An example of an action plan

Let us consider a simple example shown in Figure 6.3 of controlling the entire cloud service, e.g., by the system designer. The described elasticity requirements, `Co1`, `Co2`, and `Co3` are not conflicting, and elasticity capabilities are searched for fulfilling these requirements. Possible elasticity capabilities are, for instance, for the case the running time is higher than 10 hours and the cost is still in acceptable limits to scale-out for the computation service topology, increasing the processing speed. An example of an action plan, shown in Figure 6.3 could be `ActionPlan1=[ [increaseReplication], [scaleOut, setThreadPool=100]]`. This action plan would address performance issues for the second elasticity requirement `Co2`, and availability issues for the third elasticity requirement `Co3`. Each of the generated elasticity capabilities are mapped into complex API calls. For instance, `increaseReplication` elasticity capability would consist of calls for adding and configuring a new database node and configuring the cluster for higher replication, while the `scaleOut` elasticity capability would be the addition of a new virtual machine, deployment of the `ComputationEnd` on the new machine, and necessary calls for the new instance of the service unit to join the computation topology cluster.

6.3.2 Resolving elasticity requirements conflicts and generating action plans

We identify two types of conflicts: (i) conflicts between elasticity requirements targeting the same abstraction level, and (ii) conflicts that appear between elasticity requirements targeting different abstraction levels. For the first case, sets of conflicting constraints are identified and a new constraint overriding previous set is added to the dependency

graph for each level. In the second type of conflict the constraints from a lower level (e.g., service unit level) are translated into the higher constraint's level (e.g., service topology level), by aggregating metrics considering the dependency graph. Since the problem is reduced to same-level conflicting elasticity requirements, we use the approach for the same-level conflicting elasticity requirements and compute a new requirement from overlapping conditions.

We identify two types of conflicts: (i) conflicts between elasticity requirements targeting the same abstraction level, and (ii) conflicts that appear between elasticity requirements targeting different abstraction levels. The process of resolving the first type of conflicts is presented in Algorithm 8. Sets of conflicting constraints are identified and a new constraint overriding previous set is added to the dependency graph for each level (lines 3-10).

Algorithm 6.1: Conflict resolution on the same abstraction level

```
1 Input: graph_i - Runtime Dependency Graph
2 Output: graph_o - Single-level Non-conflicting Runtime Dependency Graph
  constraints = findAllEnabledConstraints(graph_i) for each l in
  cloudServiceAbstractionLevels do
3 |  List<List<Constraint>,Level> confConstraints=
  |  getConflictingConstraints(constraints,l)
  |  graph_i.removeConstraints(confConstraints) List<Constraint>
  |  genConstraintsLevel; for each constraintSet in confConstraints do
4 |  |  newConstraint=solve(confConstraints)
  |  |  genConstraintsLevel.add(newConstraint)
5 |  end
6 |  graph_i.addConstraints(genConstraintsLevel)
7 end
8 return graph_o = graph_i
```

In the second type of conflict (see Algorithm 10) the constraints from a lower level (i.e., service unit level) are translated into the higher constraint's level (i.e., service topology level), by aggregating metrics considering the dependency graph. Since the problem is reduced to same-level conflicting elasticity requirements, we use the approach for the same-level conflicting elasticity requirements and compute a new elasticity requirement from overlapping conditions. In both (i) and (ii) it can be the case of conflict for elasticity requirements that are targeting different metrics that influence each other (i.e., cost and availability- when availability increases, the cost increases as well). However, knowing how one metrics' evolution affects the other is a research problem itself that we envision as future work.

For a better illustration we can consider the elasticity requirements in Figure 6.4, described by the service user/owner, focusing on the execution of service unit `Hadoop Slave` given that `Hadoop Master` is finished doing the "map" part. In this case, `cons1` and `cons 2` are conflicting directives from the same level of abstraction, and we can

Algorithm 6.2: Conflict resolution on the different abstraction levels

Input: $graph_i$ - Runtime Dependency Graph
Output: $graph_o$ - Cross-level Non-conflicting Runtime Dependency Graph **for** ***each*** $l1$ *in* $cloudServiceAbstractionLevel$ **do**
  for ***each*** $l2$ *in* $cloudServiceAbstractionLevel$ **do**
    if $l1 \neq l2$ **then**
      $confConstraints$.add(findConflicts($l1$,$l2$))
    end
  end
  $graph_i$.removeConstraints($confConstraints$)
  $transl$=translateToHigherLevel($confConstraints$)
  $graph_i$.addConstraints($transl$)
end
return $graph_o$=*solveSameLevelConf*($graph_i$)

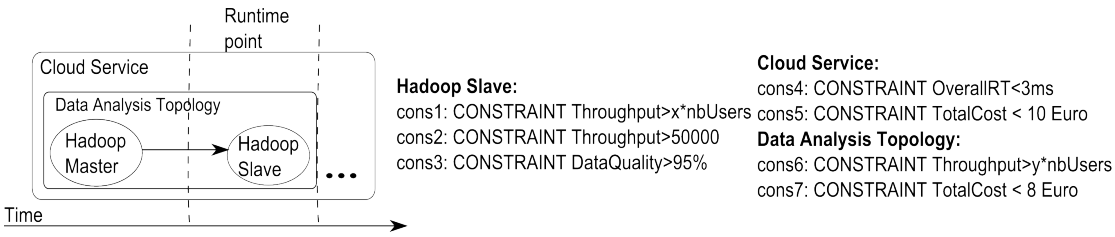


Figure 6.4: Cloud service and possible conflicting elasticity requirements

compute a new directive by choosing as maximum throughput between $x * nbUsers$ and 50000. On the other hand, we can have the second type of conflicts for the $TotalCost$ metric: if we consider that `Hadoop Master` has finished its execution with a cost of 4 Euro, fulfilling `cons7` on the service topology would be in contradiction with `cons5` since the cost is an aggregation of the cost for the service unit `Hadoop Master` and service topology `Data Analysis Topology`. For resolving the conflicts we would need to produce a constraint, `TotalCost<6`, replacing the conflicting ones.

For generating an action plan for cloud service elasticity control, we formulate the planning problem as a maximum coverage problem: we need the minimum set of capabilities that help fulfilling the maximum set of requirements. Given the current cloud service state, we can apply a number of elasticity capabilities from the *Elasticity Capability Set ECS*. As described in Equation 6.1, for each elasticity capability enforcement, we reach a state with a set of requirements fulfilled EC_x. We therefore need the minimum set of capabilities that fulfill the maximum set of requirements. Since maximum coverage problem is an NP-hard problem, and our research does not target finding the optimal

solution for it, we choose the greedy approach that offers an $1 - \frac{1}{e}$ approximation.

$$ECS = \{EC_1, EC_2, ..., EC_n\}$$
$$EC_x = \{Fulfilled(R_{x_1}), ..., Fulfilled(R_{x_y}) | R_i \in Requirements\} \quad (6.1)$$

The main step of the greedy approach consists of finding each time the elasticity capability EC_i fulfilling the most constraints and improving the most strategies. After selecting an elasticity capability in this iterative process, the ECS needs to be recomputed since the context of the service is changed and the effect of applying EC_j will be different than before applying EC_i. For now we consider that the effects of enforcing an elasticity capability are introduced by the user, our framework presented in Section 5.3.4 offering mechanisms for easily plugging-in tools that automatically detect the effect of an elasticity capability.

The greedy approach shown in Algorithm 9 takes as input the dependency graph and returns the sequence of actions necessary for enforcing the constraints. The main

Algorithm 6.3: Generating Action Plan for Constraint Enforcement

```
1 Input: G - Cloud Service Dependency Graph
2 Output: ActionPlan vConst = findViolatedConstraints(G) while size(vConstr)
  > 0 & lastFixed>0 do
3   for each l in cloudServiceAbstractionLevel do
4     selectConstraints(vConstraints,l) actionSet=evaluateEnabledActions(G,
      vConstraints) Action=findAction(actionSet) with
      max(constraintsFulfilled -
5     constraintsViolated + strategiesImproved-strategiesWorsened) Add action
      Action to ActionPlan vCons = getViolatedConstraints(G,
      estimatedEffect(Action))
6   end
7   wait(refreshTime)
8 end
9 return ActionPlan
```

step of the plan generation loop (lines 2-9) consists of finding each time the action for fulfilling the most constraints and improving the most strategies. For evaluating this, we support modeling each action with the manner in which the metrics are affected by its enforcement (i.e., scale out with `VM of type x increases the cost with 200 Euro`). Automatically determining the effect each action has upon all the relevant metrics of all the nodes of the cloud service (e.g., the effect of reconfiguring data end upon the business end) is an ongoing work. We select actions based on the requirements that are not fulfilled at the time. For instance, if a cost requirement is violated for a service topology, we prioritize the actions that we try, giving higher priority to actions that affect cost for the service topology and for the service units that compose this service topology (since that cost is being aggregated for computing service topology cost). We

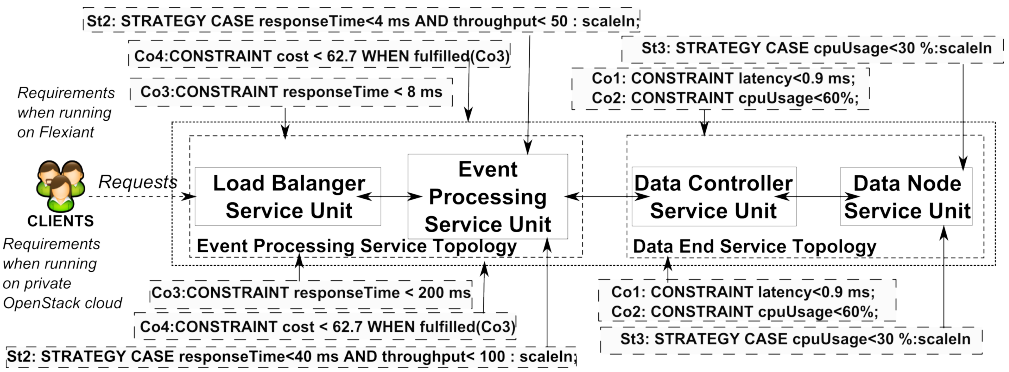


Figure 6.5: M2M DaaS with SYBL elasticity requirements

simulate the enforcement of the action by applying the described effects, and compute the number of fulfilled constraints through action enforcement as the difference between the number of constraints that are fulfilled and the number of constraints violated. For instance, increasing replication with a factor of one from the example above would affect negatively cost and positively performance and availability, resulting in the number of fulfilled constraints through action enforcement equal to one if cost requirement is violated and two otherwise.

6.3.3 Enforcing action plans

For controlling the elasticity of cloud services, tools monitoring the elasticity and the different types of metrics targeted by the cloud service user are necessary. Although at the moment existing cloud APIs offer only access to low-level resources, elasticity control of cloud services would also impose the existence of cloud APIs that take into account the different levels of metrics or the cloud service structure.

For overcoming this situation, we use MELA framework that aggregates low-level metrics for achieving higher level ones, and use existent resource-level control capabilities for manipulating higher level quality and cost. For instance, the cost of a service unit would be composed of the different types of cost associated to each resource associated with the service unit, like cost depending on the number of virtual machines, cost for intra-unit communication, or I/O cost. The cloud service cost is computed as the sum of service unit cost, inter-unit communication cost, and possible licensing costs.

Considering long running services, the stakeholders can evaluate the actions generated, and revise their requirements on the basis of application behavior. This is possible either before or after action plan enforcement, rSYBL re-running the elasticity control loop, starting from the first step described in Section 6.3.1.

Setting	Flexiant	OpenStack
Small instance GB:CPU	2GB:2CPU	1GB:1CPU
Small instance price	6 Flexiant Units/h	3 OpenStack Units/h
Network interface card	0.13 Flexiant Units/h	0.13 OpenStack Units/h

Table 6.1: Experiment settings

6.4 Experiments

6.4.1 Controlling elasticity with rSYBL: M2M DaaS Cloud Service

Considering the machine-to-machine (M2M) DaaS presented in Chapter 4.2, we simulate clients that send sensor information (i.e., create artificial workload for the Local Processing Service Topology). Specifically, the M2M DaaS is comprised of an *Event Processing Service Topology* and a *Data End Service Topology*. The *Event Processing Service Topology* consists of a *Load Balancer Service Unit* and an *Event Processing Service Unit*, that analyzes and stores data in a NoSQL cluster, in this case Cassandra-based, composed of a *Data Controller Service Unit* (i.e., Cassandra seed) and a *Data Node Service Unit* (i.e., Cassandra node).

We deploy the M2M DaaS service on two different cloud infrastructures: (i) using Flexiant[2] cloud provider, we deploy on their public cloud, and (ii) on our OpenStack[1]-based private cloud. We simulate data sensors that send information to *Event Processing Service Topology* with a python-based load generator that sends random data to our M2M DaaS. As the two clouds considered are different, they differ in reliability, estimated cost, and in quality characteristics, even when using similar resources. Table 6.1 shows the `settings of our experiments` in terms of estimated cost. The Flexiant cloud provider costs vary with the user type, and it is manually set by Flexiant cloud administrators. For our case, the price for an instance with 2 GB and 2 CPU is 6 units per hour, where the units can be bought at varying prices (from 11£ per 1000 units to 4700£ per 500000 units). From OpenStack we select an `m1.small` instance, with 1 GB and 1 CPU, and compute an equivalent cost of half the number of units from Flexiant (based on our assumption that OpenStack private cloud units are much cheaper than Flexiant units due to maintenance costs, e.g., electricity, or administration). The price of a network interface card, associated by default with each instance is 0.13 units, that we also set for OpenStack cloud experiment settings.

The SYBL elasticity requirements are associated with the M2M DaaS at different levels (e.g., cloud service level, service topology level). Since the cloud infrastructures are different, the requirements have to be adjusted for the providers, as they provide different performance at different costs. For Flexiant, we set a requirement of keeping response time less than 8 ms (see `Co3` for Flexiant case), while for the OpenStack private cloud we set the requirement of maintaining response time below the limit of 200 ms (see `Co3` for OpenStack case). As we have equated the costs for the two providers considering resources provided, we maintain the same cost requirement for the two cases (see `Co4`).

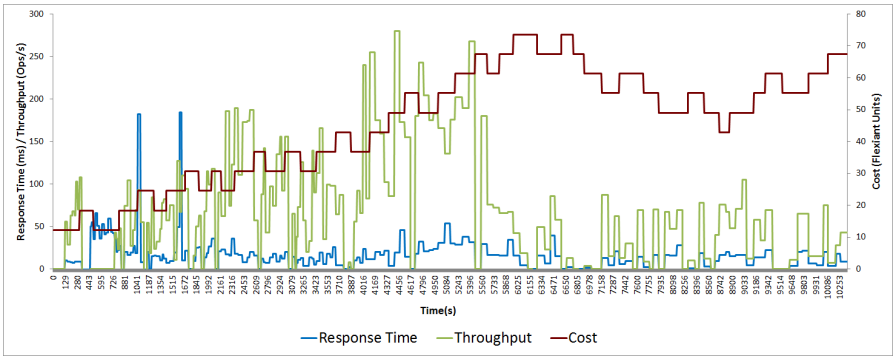


Figure 6.6: Event Processing Service Topology on Flexiant public cloud

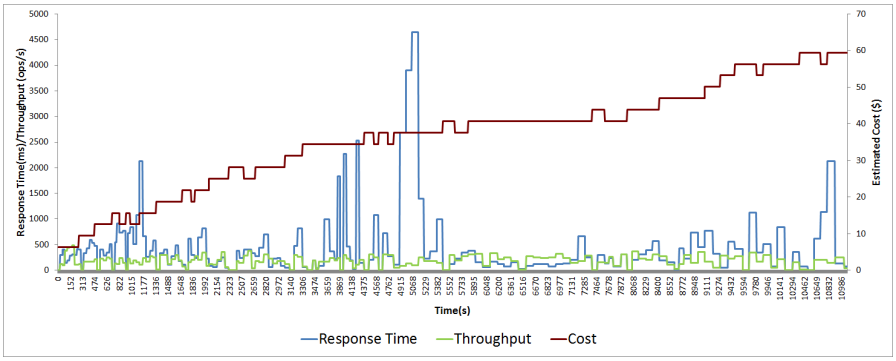


Figure 6.7: Event Processing Service Topology on OpenStack-based private cloud

Figure 6.6 shows how the `Event Processing Service Topology` of the DaaS cloud service is affected by rSYBL control actions on the Flexiant cloud. An action enforcement is reflected in a change of cost, the deployment of a new instance with the necessary configurations being reflected in a cost increase, while the removal of an instance associated to a unit being reflected in a decrease in cost. We can see that starting from minimal deployment configuration (1 VM per service unit), rSYBL manages to find a level of resources configurations where any control enforcements do not affect the quality characteristics as it was the case in the first part of the experiment, when the response time has a short peak of 180 ms. For the second case presented in Figure 6.7, running on OpenStack cloud, the evolution of the service is different since the cost (i.e., the actual price that needs to be paid at the end of day) is smaller, while the performance (e.g., response time for the `Event Processing Topology`) is as well smaller, rSYBL having to allocate a lot of resources.

Listing 6.1: Example of elasticity requirements in XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CloudService id="CloudService">
  <ServiceTopology id="DataEndServiceTopology">
```

```
        <SYBLDirective Constraints="Co1:CONSTRAINT latency <0.5
            ms;
          Co2:CONSTRAINT cpuUsage &lt; 83 %" />
            <ServiceUnit id="DataControllerServiceUnit" />
            <ServiceUnit id="DataNodeServiceUnit" >
                  <SYBLDirective Strategies="St3:STRATEGY CASE
                cpuUsage < 40 % : scalein" />
            </ServiceUnit>
  </ServiceTopology>
  <ServiceTopology id="EventProcessingServiceTopology">
          <SYBLDirective Constraints="Co3:CONSTRAINT
              responseTime
            < 350 ms" Strategies="St1:STRATEGY CASE
            cost < 0.5 : maximize(throughput)"  />
          <ServiceUnit id="LoadBalancerServiceUnit" />
          <ServiceUnit id="EventProcessingServiceUnit" >
                  <SYBLDirective Strategies="St2:STRATEGY CASE
                responseTime < 360 ms AND throughput < 400 :
                scalein" />
          </ServiceUnit>
  </ServiceTopology>
</CloudService>
```

The results of running a sinusoidal workload with a high amount of bursts are shown in Figure 6.8 and Figure 6.9. For this case, we assume the cost of a VM 0.12$ per hour. We can see that for the `Event Processing Topology` the response time decreases and throughput increases when new instances of `Event Processing Service Unit` are added, while the reverse is happening when instances are removed. Since for now the control algorithm is a reactive one, and we don't have a structured behavioral information on the application, actions are not always guaranteed to be the best ones. However, we envision as future work comparing different control algorithms, and modeling and using cloud service behavior information. For the `Event Processing Topology`, control actions are enforced whenever the constraint `Co3` is violated, or the strategies `St1` and `St2` are decided to be enforced due to their triggering conditions being true.

Figure 6.9 shows how the cost per client per hour evolves compared to the number of clients. The cost per client per hour is a complex metric, which tells the elasticity controller and the cloud service developers or providers how efficiently resources are allocated to the entire cloud service. We can see that the cost reported to the number of clients (Cost/Clients/h), on the left axis in Figure 6.9 is kept stable, while the number of clients increases. This means that the cloud service is elastic, managing to accommodating a varying number of clients. Towards the end of the workload, when the number of clients is dropping, we can see a slight increase of Cost/Client/h, which means that even though we don't have that many clients, the application still has some re-balancing processes going on, so rSYBL is not yet able to completely scale back to expected cost for this

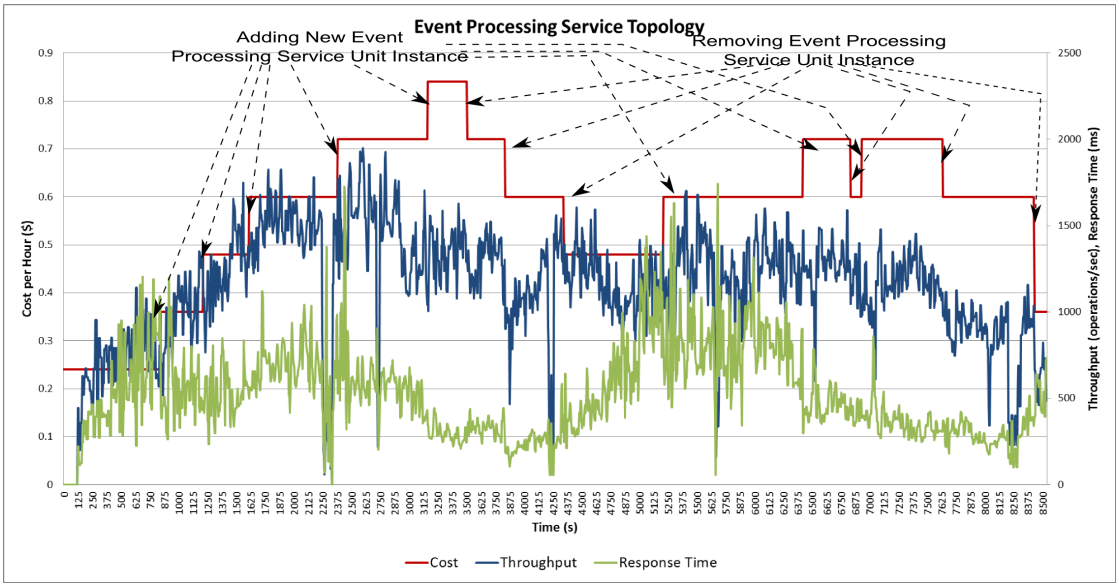


Figure 6.8: Elasticity Evolution of Event Processing Service Topology of the M2M Cloud Service: throughput versus cost on Event Processing Service Topology

number of clients.

6.4.2 Analyzing multiple levels of control: YCSB+Cassandra Cloud Service

In the second part of our experiment, we use a cloud service with two service topologies: one made from YCSB[4] Cassandra clients, the second one being a Cassandra[5] cluster, with two types of service units: Cassandra Seed, the unit acting as the controller of the cluster (i.e., Data Controller Service Unit), and Cassandra Node (i.e., Data Node Service Unit). We experiment taking different level of control actions for the Cassandra NoSQL cluster. For the current experiment, the number of actions available is limited to scaling in and out at service unit level and at service topology level, by adding/removing virtual machines hosting data nodes, or by instantiating entire new data clusters, and making the proper configurations for them to receive requests from the YCSB clients.

Conflicting directives resolution: For overcoming situations where multiple users of the same service specify elasticity directives we identify conflicting directives and compute new directives or even eliminate directives in case of complete contradiction. For the current prototype, we have implemented only contradicting directives resolution by finding directives from different levels referring to the same metric that are in contradiction with one another. So, for instance, for the current example constraint "`Co1`" specifying that the CPU usage should be greater than 90% is in contradiction with both the underlying

[4] `https://github.com/brianfrankcooper/YCSB`
[5] `http://cassandra.apache.org/`

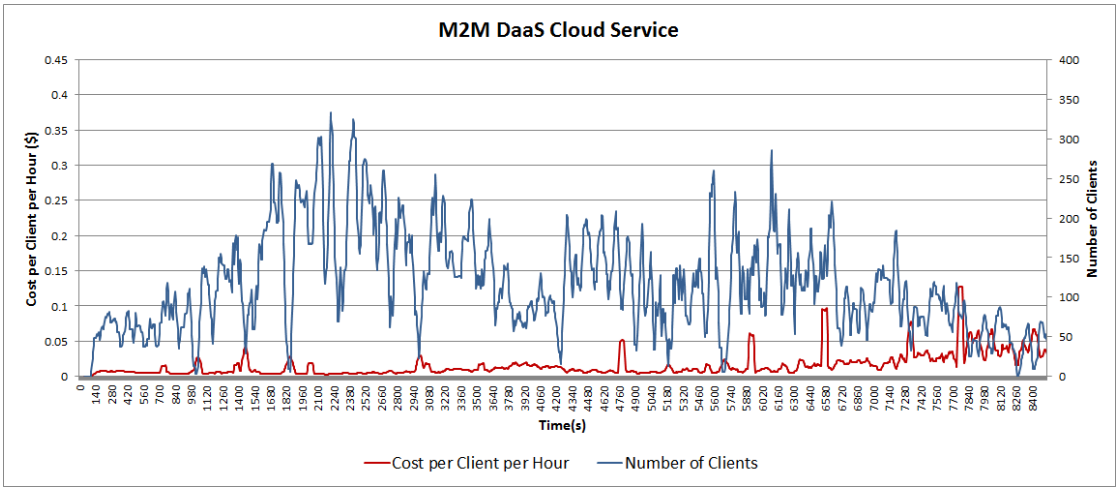


Figure 6.9: Elasticity evolution of cloud service: cost-per-client-per-hour versus throughput

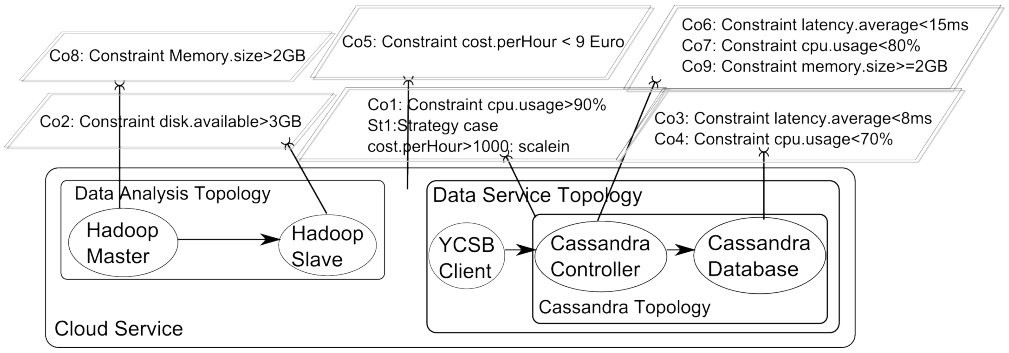


Figure 6.10: Cloud service structure and elasticity directives

constraints ("Co4" and "Co7") that specify that the CPU usage should be less than 70%, respectively 80%. This is true for the fact that CPU usage at service topology level is aggregated only from CPU usage at lower levels, leading to an impossible situation where the average needs to be higher than a value, and each of the items of the average are less than that value. Constraint "Co1" is eliminated because it has the highest number of directives contradicting with it. After that, the strategies are enforced, and the elasticity control engine finds the actions healing the violated constraints.

For reflecting the importance of higher level elasticity control in addition to the obvious low level one, Table 6.2 presents performance and cost data on different Data Service Topology configurations. We assume each virtual machine costs 1 EUR/hour. The first important reason for enabling topology level elasticity is that multiple clusters remove the single-point of failure problem, decreasing the probability of failures that is imminent for the case of highly intensive workloads with a single Data Controller Service

Config.	DB Controllers	DB Nodes	Total execution time	Cost
Config1	1	3	578.4 s	0.48
Config2	1	6	472.1 s	0.91
Config3	2	2	382.4 s	0.42
Config4	3	7	372.2 s	0.72

Table 6.2: Cost and execution time for Data Service Topology units

Config.	DB Controllers	DB Nodes	Workload	Total execution time	Cost (Units)
Config1	1	3	Workload1	44 min	9.13
Config3	2	2	Workload1	28.4 min	5.88
Config1	1	3	Workload2	>3h+connection failures	> 37.56
Config3	2	2	Workload2	102.75 min	21.53

Table 6.3: Cost and execution time: comparison on different workloads

Unit.

To show the importance of higher level elasticity control, Table 6.3 presents performance and cost data on different Data Service Topology configurations and different workloads. We use two update-heavy YCSB workloads[6], the `Workload 1` having ten times less operations to be executed than `Workload 2`. We assume the OpenStack costs in experiment above (see Table 6.1).The first important reason for enabling topology level elasticity is that multiple clusters remove the single-point of failure problem, decreasing the probability of failures, imminent for the case of highly intensive workloads with a single Data Controller Service Unit. We show how 2 clusters (Config 3) can decrease the final cost as opposed to a single cluster (Config 1), and more importantly that it can avoid errors due to overloading. For instance, for the more intensive and longer workload, `Workload 2`, a single Cassandra cluster, although with multiple virtual machines for the slave component, reaches a point where it cannot serve requests anymore, when only the service unit level control is enabled.

Therefore, enabling various types of actions, and creating controllers that differentiate among them taking into consideration the effect they have not only on the current part of the cloud service that is being reconfigured, but on the overall cloud service and on various other parts as well, can greatly improve cloud services elasticity. With the capability to control service's elasticity both at service unit level and at topology level, rSYBL can improve the elasticity control of cloud services both from the performance and from the cost perspective.

Figure 6.11 shows how the elasticity control engine can scale the Data Service Topology both at service unit and at service topology level, when directives shown in Figure 6.10 require such actions. Considering an initial configuration of a single cluster containing one controller and one simple node, the elasticity control engine takes firstly a scale out

[6] `https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads`

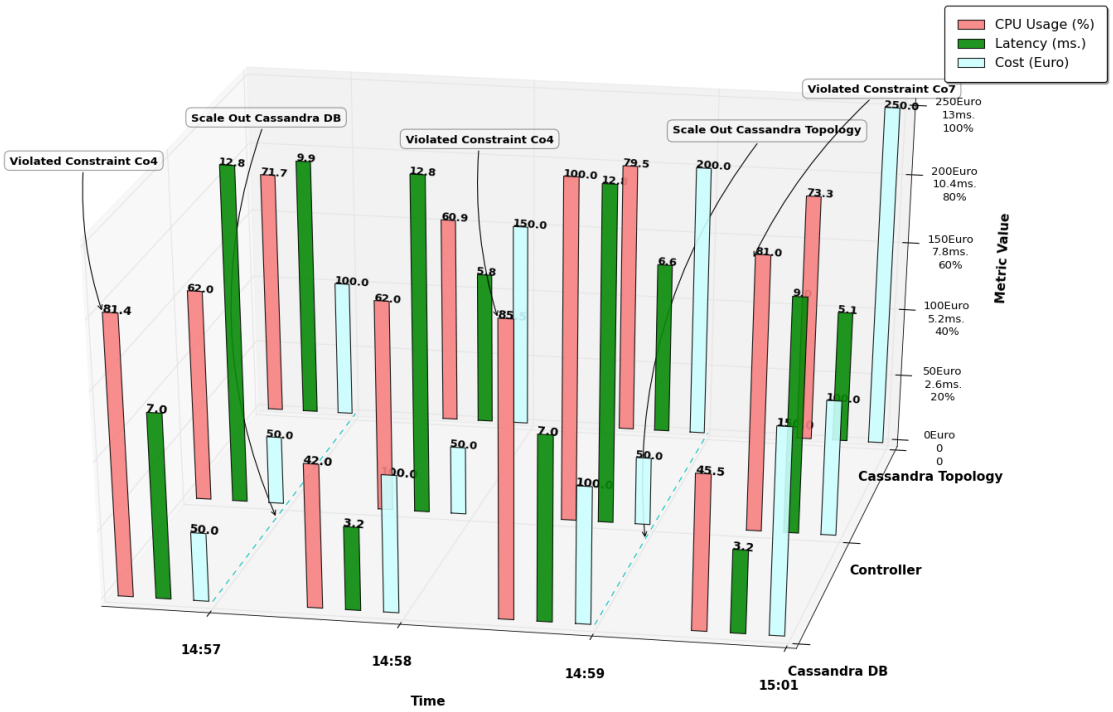


Figure 6.11: Metrics (CPU usage, cost and latency) and elasticity actions for service units in Data End Service Topology

action at service unit level for fixing the broken constraint "Co4". After that, it takes a new scale out action for topology level, due to the violated constraints "Co4" and "Co7".

6.4.3 rSYBL performance analysis

rSYBL needs a dedicated VM, but can be collocated with other service management tools like MELA. The overhead of running rSYBL is $Cost_{VM} + Cost_{Network\_eGress}$, where $Cost_{VM}$ is the cost of a VM for current cloud provider, given that we know approximately how long we would like the service hosted, and we can choose a subscription-based cost schema for this VM, and the cost of communicating among regions belonging to the same cloud provider $Cost_{Network\_eGress}$, for the case we have a multi-region deployment. For the stable workload case, this cost is not justified since we have no need for elasticity, as we can create the optimum static configuration and use it. In the case the workload is variable, it makes sense to try to reduce the bigger cost that usually are connected with virtual machines and storage disks. Moreover, as rSYBL also enables adaptive re-configuration of different service units, or service topologies, depending on the workload and elasticity requirements, rSYBL control can produce even better results than using the over-provisioning strategy.

Controlling the service by using rSYBL empowers the user to specify the requirements s/he is interested in, at the level and for the unit that fits best, since most of the times,

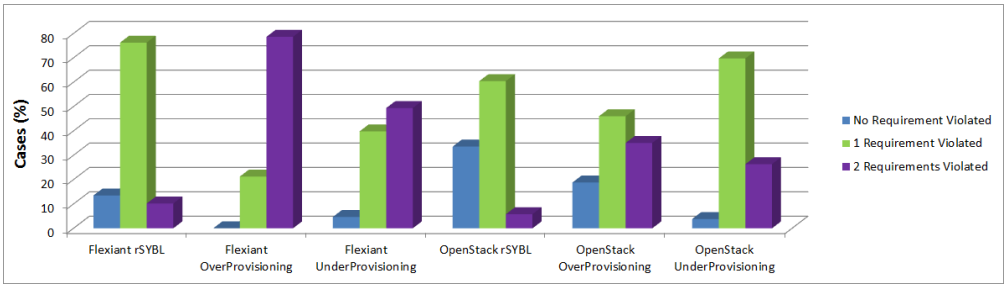


Figure 6.12: Requirements fulfillment on Flexiant and OpenStack

the user knows best what is his/her budget, and what are is the desired quality. As rSYBL's main goal is fulfilling user's requirements, we analyze the degree with that rSYBL manages to fulfill user requirements, for the M2M DaaS described in Section 6.4.1. We compare rSYBL control outcome with two stable cases that are manually configured for this experiment: (i) under-provisioning strategy (fixed configuration with minimum resources used with rSYBL - 4 VMs), and (ii) over-provisioning strategy (fixed configuration with maximum resources used with rSYBL - 14 VMs in Flexiant and 17 VMs in OpenStack). When computing the cost for the case of running with rSYBL control strategy, we factor in the cost for rSYBL to run as part of the M2M DaaS cost. We want to understand whether and how much the elasticity performance impact affects requirements (i.e., each control action initially decreases performance, and needs a 'cool-down period' [49]).

Analyzing the comparison from Figure 6.12, we can see that rSYBL is better than both under-provisioning and over-provisioning strategies, on both cloud providers used. In the over-provisioning case, while most of the times the response time requirement is fulfilled, the one on cost (`Co4`) and the one on data end CPU usage are not fulfilled (`Co2`), due to the fact that we have a continuously high number of resources for the `Event Processing Service Topology`, for which the maximum resources of the `Data End Service Topology` allocated by rSYBL in Section 6.4.1 are not enough. For the current framework, rSYBL takes only reactive actions, reason for which the cases in which one requirement is violated is quite large. We see as future work incorporating into rSYBL predictive decision making, thus decreasing the number of cases in which requirements are violated.

CHAPTER 7

A Complex Use-Case for rSYBL Elasticity Controller: Heterogeneous Control for Cloud Services

This chapter focuses on a complex use case, consisting of a service deployed on multiple heterogeneous clouds. Challenges are highlighted, and rSYBL and its model are extended for being able to control the elasticity of complex services deployed in multi-cloud environments.

7.1 Overview

A considerable amount of work has been put into cloud service control, focusing on controlling cloud services of various types, or meeting various types of stakeholder requirements [3, 94, 106]. Next to these challenges, some services might need to be distributed in several types of clouds, each with different characteristics (e.g., mini-cloud for managing smart buildings in combination with a public cloud). Within this multi-cloud configuration, the cloud service elasticity should be controlled as a whole, for fulfilling stakeholder (e.g., service developer or service provider) requirements, possibly at multiple abstraction levels of the service.

In this case, several challenges need to be tackled when designing the control for the services executed across multiple clouds. First of all, the *programming interfaces* offered by various cloud providers can differ both from the point of view of the services offered, and from the perspective of the protocols used (e.g., Flexiant's FCO[1] REST API, or OpenStack[2] python or java-based libraries). Secondly, the *run-time control capabilities* offered by different clouds have different enforcement time and results (e.g., in mini-clouds

a VM spawning action usually takes much longer than in a public cloud). In this context, the user needs an elasticity control with an *end-to-end view of the multi-cloud service*, understanding different levels of service abstraction, as opposed to current control flows where the user needs to deploy separate controllers on each cloud, and treat the distributed service parts as different services. A controller offering this end-to-end control perspective has to analyze various information types characterizing the multi-cloud environment, and evaluate the impact upon the rest of the service when enforcing an action. For instance, in the context of smart city services, when releasing virtual resources in a smart building cloud, e.g., in sensors' low-activity period, a controller should consider the impact that this would have on the rest of the service possibly deployed on a public cloud.

To address challenges above, we propose mechanisms of controlling the service from an end-to-end perspective, transparent to users from the point of view of the clouds heterogeneity. To this end, we model details specific to multi-cloud deployments in order to base our mechanisms on a common representation of the highly heterogeneous services offered by cloud providers. For supporting controlling services from an end-to-end perspective, we focus on modeling relationships among multi-cloud distributed service units, to be used as a basis for elasticity control of the cloud service.

The contributions presented in this chapter are the following: (i) a model for multi-cloud deployed services, with focus on relationships that occur among service parts, (ii) new mechanisms for relationship-driven control of multi-cloud services.

We extend our rSYBL [24] controller with proposed multi-cloud mechanisms, and show that by specifying simple, high level requirements, stakeholders can have elastic services deployed on multiple, heterogeneous clouds controlled at runtime with rSYBL elasticity controller. Moreover, we emphasize the extensibility of our framework that can be easily customized to support a variety of clouds and enforcement APIs.

7.2 Motivation, Background and Related Work

7.2.1 Motivation

Let us consider a Machine-to-Machine Data as a Service (M2M DaaS), for a smart city. The smart city has millions of sensors in each building, and a group of buildings sending their data to a local mini-cloud (e.g., NuvlaBox[3], or Ubuntu Orange Box[4]) in which the M2M local processing units are deployed and to which the sensors send data (left side of Figure 9.1). The data analyzed locally is sent to a public cloud containing the rest of the M2M service (right side of Figure 9.1) for the higher availability of public clouds and due to the fact that smart building mini-clouds can store a limited amount of data. The M2M service needs to be controlled both in the local mini-clouds and in the public one, in order to obtain the best possible performance at the best possible cost. To this end, user's

[1] `http://www.flexiant.com/flexiant-cloud-orchestrator/`
[2] `http://developer.openstack.org/`
[3] http://sixsq.com/products/nuvlabox.html
[4] http://www.ubuntu.com/cloud/tools/jumpstart-training

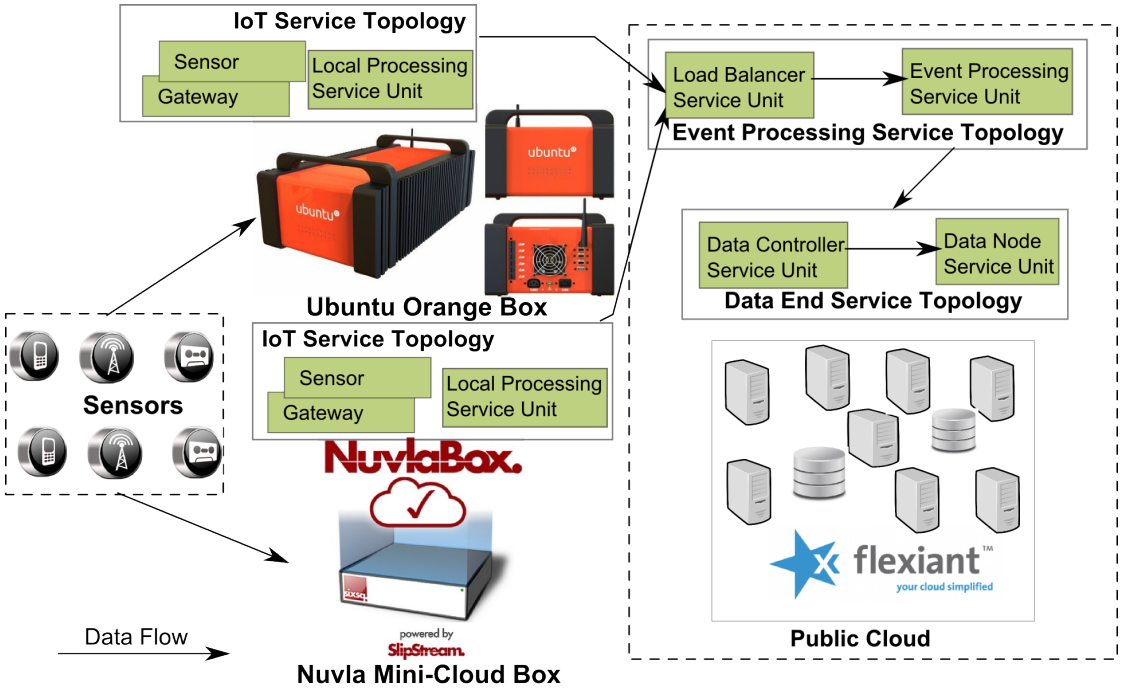


Figure 7.1: Motivating scenario

(i.e., any service stakeholder) requirements for an elasticity controller play a major role, stating the trade-off with regard to cost, quality and performance. We envision that the users would describe requirements in a cloud agnostic manner, at a high level, considering application level elasticity metrics, and all the rest is being handled by the controller. Given that the M2M service is deployed on multiple clouds, controllers would need to use various programming interfaces and service elasticity capabilities offered by different types of cloud providers. Moreover, due to the fact that users need an end-to-end control of the service, the controller has to understand the relationships among service parts deployed on different clouds, and control them accordingly.

To address above-mentioned challenges, we propose a model representing multi-cloud service information for controllers to understand elasticity capabilities specific to different cloud providers. Based on this model, we design control mechanisms using common methods of enforcing control capabilities, and focusing on relationships existent in multi-cloud services. Having as central focus service stakeholders, we are not interested in providing standardized access to federated clouds, but quite the opposite: we want to help them profit from the current heterogeneity for creating multi-cloud elastic services within nowadays dynamic cloud context. To achieve this, the provider/developer should be relinquished from the burden of dealing with cloud-specific control, or with resource models on each cloud (e.g., gateways control versus normal virtual machine control).

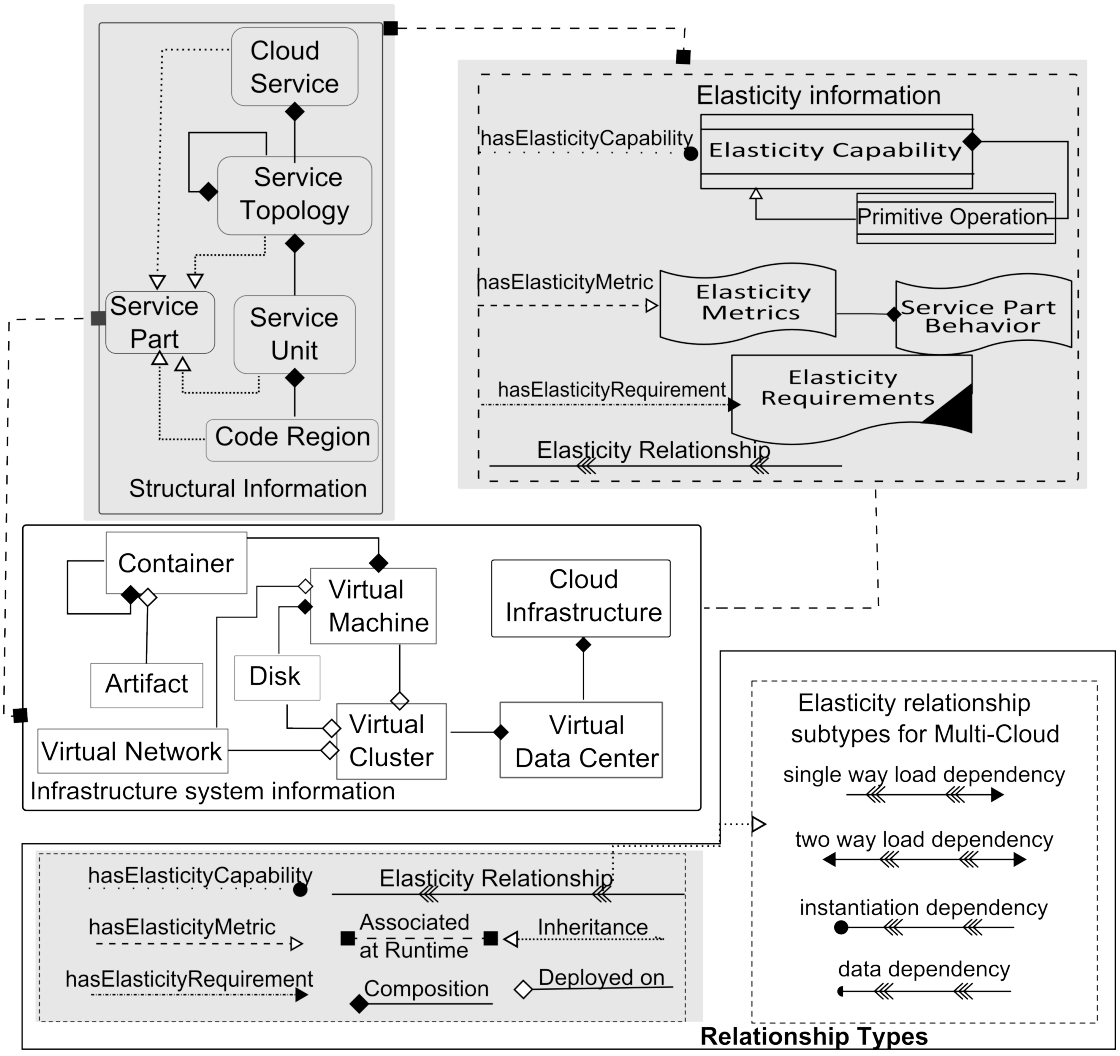


Figure 7.2: Cloud service model

7.3 Multi-cloud elasticity control

For enabling the multi-cloud control described in our motivation scenario, the controller needs to analyze multi-cloud service information, from capabilities of each service part, each infrastructure element, to each measured metric. For instance, for the case of the M2M DaaS service in Figure 9.1, gateways' elasticity capabilities and deployment structures from mini-clouds would differ from deployment stacks and elasticity capabilities of web services deployed in the public cloud. Therefore, clouds heterogeneity results in a diversity of the cloud services, not only of the supporting infrastructures.

This reveals a two-fold challenge: (i) from a conceptual perspective, we need to enable the user to have an end-to-end view of the service, (ii) from a technical point

of view, the controller has to understand a variety of primitives and protocols from a variety of providers. For addressing the first challenge, we model the multi-cloud specific information by extending our model for single-cloud services [24]. We address the second challenge in Section 7.3.2, showing how we use the modeled information to enable dependency-aware multi-cloud service control and how we abstract elasticity capabilities from primitive operations that are available for the different virtual resources offered by cloud providers.

7.3.1 Multi-cloud Service Model for Elasticity Control

In our control approach described in Section 9.2, the cloud service is modeled together with the infrastructure in which it runs, that is single-cloud. For *multi-cloud service control*, we extend the model used in the single cloud control (Figure 7.2, gray background) for being able to better represent the infrastructure, from virtual machine to software artifacts and their placement and configuration (Figure 7.2, white background).

For understanding the cloud infrastructure used by the service during runtime, and the different resources from different clouds that are used to host the service's software stack, we introduce the following virtual infrastructure concepts:

- *Virtual Data Center* – the data center where the cloud service is deployed (e.g., Amazon EU location 1)
- *Virtual Cluster* – a group of resources physically isolated from the rest of the resources (e.g., the resources for a specific company)
- *Disk* – a disk instance associated to a VM, or shared among multiple VMs
- *Virtual Network* – network associated to a virtual cluster
- *Container* – any type of software having the property of being used by others as a container (e.g., Docker[5], web server, gateway, or data store).
- *Artifact* – any type of atomic software (e.g., web service, sensor, or data set).

Each of the service parts described in the structural information (upper left side of Figure 7.2) have associated elasticity information (upper right side of Figure 7.2), such as *Elasticity Metrics*, *Elasticity Requirements*, or *Elasticity Capabilities*. The latter represents complex actions exposed by service parts, composed of *Elasticity Capabilities* exposed by infrastructure elements described above, to which we refer as *Primitive Operations*. We distinguish among them since primitive operations can be linked to an actual operation made by the elasticity controller, e.g., API call, while elasticity capabilities are more abstract actions that are composed of several primitive operations (e.g., a scale in elasticity capability for the service unit level can be a decommissioning primary operation achieved by the software/artifact level and a remove VM primary

[5]http://www.docker.com/

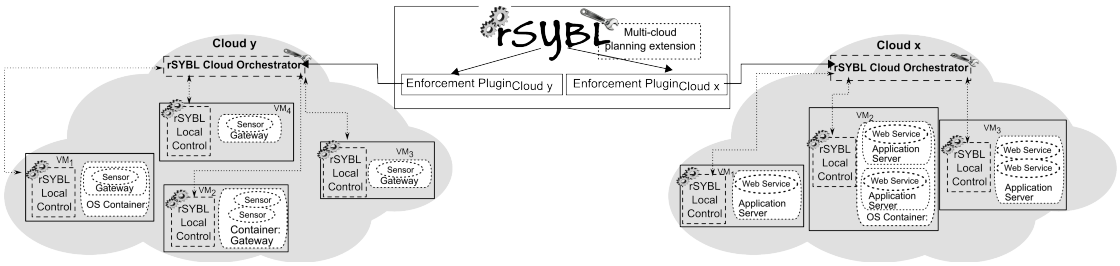


Figure 7.3: rSYBL multi-cloud control framework

operation achieved at the virtual machine level). Enforcing elasticity capabilities is equivalent to the enforcement of the associated primitive operations, considering their dependencies.

For understanding how different service parts interact, we introduce the following elasticity relationship types (bottom of Figure 7.2), as subtypes of *Elasticity Relationships*:

- *Single way load dependency* – a change in the antecedent load causes a similar change in consequent load
- *Two way load dependency* – a change in the antecedent or consequent causes a similar change in the other
- *Instantiation dependency* – for the instantiation of the consequent the antecedent should to exist. This relationship can also contain other properties, like the data needed to be transferred among the two.
- *Data dependency* – the specified data should to be transferred among the antecedent and the consequent.

Relationship concepts above are used in order to connect two parts of the service, regardless on whether or not they are in the same cloud. All the above concepts are represented at runtime through a runtime dependency graph, which has as nodes the concepts and as edges the relationships presented in the model (bottom of Figure 7.2), and that is used by the elasticity controller to take control decisions. We obtain relationships above from service profilers that have analyzed the execution of respective services [90], or from other various service stakeholders. The model contains sufficient information for the controller to understand the complex implications of an enforcement of an elasticity capability on one end, on the rest of the cloud service hosted in other cloud infrastructures.

7.3.2 Multi-cloud Service Elasticity Control

Based on the model above, we propose mechanisms to control the multi-cloud service, based on two major issues that rise out of the multi-cloud setting: (i) mechanisms to control service parts relationships/dependencies in multi-cloud (Section 7.3.2), and (ii) mechanisms to manage primary operations' heterogeneity (Section 5).

Control mechanisms based on service part dependencies in multi-clouds

For the elasticity control of a multiple heterogeneous clouds deployed service, we evaluate the runtime dependency graph modeling the various types of information (e.g., structural, elasticity, or infrastructure) with regard to the cloud service. The major difference in the control mechanism rests in the way in which we evaluate the actions to be enforced. For the multi-cloud scenario, we evaluate the consequences of the enforcement of one elasticity capability on the rest of the parts of the service, possibly deployed in other cloud infrastructures.

Algorithm 7.1: Multi-cloud elasticity capability analysis

```
1 Input: evalEC - evaluated elasticity capability, sp - targeted service part, graph -
  current dependency graph
2 Output: resultedECs - Elasticity capabilities to be enforced with evalEC
  initialReq=evaluateRequirements(graph) initECs =
  evaluateInstantiations(evalEC,sp,graph) for each service_part in spsToEval
  ecs.add(node.evaluateECs()) simulateEnforcementOfECs(initECs)
  spsToEval.add(simulateImpact(graph.getDataRel()))
  spsToEval.add(simulateImpact(graph.getLoadRel()))
  simulateEnforcementOfECs(ecs) if
  evaluateEnforcement(dependencyGraph,initialReq) then
3 |  resultedECs.add(initECs) resultedECs.add(ecs)
4 end
5 return resultedECs
```

Algorithm 5 shows the evaluation procedure for an elasticity capability considered by our control mechanism. When choosing an elasticity capability to be enforced, as part of the control mechanism, we evaluate the elasticity relationships associated with the target service part. We determine the instantiations necessary with the enforcement of *evalEC* (Line 2-3 of Algorithm 5) and simulate their enforcement on *graph* for preparing it for the other relationship evaluations. For the simulations we construct a new dependency graph, in which the implications of the relationships hold. We use *simulateImpact* function to simulate on the dependency graph *graph* the impact reflected by elasticity relationships (Lines 4-6 of Algorithm 5), by modifying metric values or used resources accordingly. We analyze the dependency graph and evaluate whether compensation elasticity capabilities should be enforced for overcoming the effect produced by our initial capability. At the end of this process we have a list of elasticity capabilities *resultedECs* to be enforced for fulfilling user's elasticity requirements, which results in an end-to-end control of the multi-cloud service.

Heterogeneous services control

Considering we have a service deployed across a heterogeneous multi-cloud environment, services offered by providers are highly diverse, both in their structure and in the elasticity

capabilities offered. As shown in Figure 7.3, rSYBL can control several types of objects, in different software stacks from gateways and sensors to Docker based stack with several web server containers containing several web services, which expose a variety of capabilities as shown in Table 7.1.

Virtual infrastructure element	Elasticity capabilities
Bash Process	change priority, kill process
Gateway	create, delete, add data repository
RabbitMQ	create, delete, modify policies, change environment variables
Data Repository	create, delete, change access rights, create new sub-repository
Tomcat	create, delete, tomcat manager-based runtime deployment and config
Docker	run new image/artifact
Disk	resize, attach
Virtual Machine	create, delete, change number of cores, change RAM

Table 7.1: Examples of elasticity primitive operations

Table 7.1 shows different elasticity primitive operations associated with various artifacts and virtual resources, which can be used simultaneously in a multi-cloud deployment. An elasticity primitive operation can be seen as an atomic operation to be executed on an infrastructure-related or software related element (e.g., virtual resource, or artifact). As we can see in this case, the nature of these primitives is quite diverse, from simple creation/deletion, to reconfigurations, or policy changing, different types of artifacts having different manner of enforcing each primitive.

After the deployment of the service on the cloud infrastructures is done, the deployment stakeholder (e.g., the deployment tool, or the developer doing a manual deployment) describes the elasticity capabilities of each service unit and service topology, as a sequence of primitive operations from the ones previously defined and the already existent default ones. These abstract elasticity capabilities associated to service parts are enforced as ACID transactions, with roll-back of the already enforced primitives if one of them fails. For ensuring the ACID property of an elasticity capability, its enforcement is first prepared through the evaluation of the enforcement possibility for each of the primitive operations that compose the capability. Moreover, the service developer can specify relations among primitives, due to the fact that in some cases a preparation of the main primitive operation is necessary, e.g., in scale in actions, a decommissioning is necessary in order to preserve the information from the disappearing virtual resource.

Infrastructure Element	Primitive	Parameters
OpenStack VM	create/remove	IP
Flexiant VM	create/remove	UUID
Flexiant Nic	create/remove/attach	UUID
HAProxy	leave/join load balancer	IP, Load Balancer Config
Cassandra	leave/join cluster	IP, Data Controller Config

Table 7.2: Currently supported primitives

7.4 Prototype and Experiments

7.4.1 Prototype

We implement the above mechanisms as an rSYBL [24] extension, for managing the interaction simultaneously with different types of clouds, which host different types of artifacts. The multi-cloud rSYBL controller is able to control, based on the dependency graph that follows the new model described in Section 7.3.1, cloud services composed of service parts distributed over multiple, heterogeneous clouds.

Figure 7.3 shows the multi-cloud *rSYBL* elasticity controller, having components deployment over different clouds, and their communication. Whenever a direct communication is not possible (e.g., a private cloud API is not publicly accessible), we deploy an *rSYBL Cloud Orchestrator* to which the cloud API calls and communication with *Local rSYBL Controller* are delegated. Although the *rSYBL* controller receives all the elasticity requirements, some are delegated to the *rSYBL Local Controller* in case the user defines local (e.g., code region) requirements, most of the times they would refer local metrics and local enforcement mechanisms (e.g., manage thread pool). Given the service description given by the service stakeholder, including its structure, the possibility of combining different artifacts and containers, and the capabilities for each artifact and container, and the monitoring information, rSYBL evaluates the dependency graph and generates a sequence of elasticity capabilities to be enforced.

The rSYBL framework currently supports a set of default services, with their primitive operations: OpenStack[6] private clouds, Flexiant[7] public cloud infrastructures, Salsa orchestration and configuration tool part[8], Cassandra[9] NoSQL database, HAProxy[10] load balancer. To this, the service developer can add their own service primitives, by adding a description of those primitives on the primitives configuration file, and implementing an rSYBL plugin that should be called when those primitives need to be enforced.

7.4.2 Experimental application end-to-end view

For our experiments we use the M2M DaaS, presented in Figure 9.1. We simulate the mini-clouds gateways (in the left part of Figure 9.1) through virtual machines

[6] http://www.openstack.org/
[7] http://www.flexiant.com/
[8] http://github.com/tuwiendsg/SALSA
[9] http://cassandra.apache.org/
[10] http://www.haproxy.org/

deployed on our private OpenStack cloud, containing sensor open data stores with sensor information from police cars. The sensors send data to an ActiveMQ[11] queue, which our `LocalProcessingUnit` evaluates and decides whether is urgent to send it. The data is either sent on an on-demand basis, or as bulk in periodic intervals, to the `EventProcessingTopology`, composed of a `LoadBalancerUnit` and an `EventProcessingUnit`, which further analyzes and stores the data in a `DataEndTopology`, composed of a `DataControllerUnit` and a `DataNodeUnit`. The `EventProcessingTopology` and `DataEndTopology` are deployed on Flexiant's public cloud infrastructure. The two clouds are heterogeneous, providing different services, different control primitives, and interaction protocols.

Within this setting, we use our multi-cloud rSYBL controller to manage the M2M DaaS deployed over the two cloud infrastructures. The elasticity capabilities available for the M2M DaaS service parts are *scale in* and *scale out*, each being composed of a different sequence of primitives from the ones specified in Table 7.2 (e.g., scale in for `EventProcessingUnit` is composed of a "leave load balancer" for HAProxy, followed by a removal of Nic and a removal VM Flexiant primitive). For control primitives enforcement in the two cloud infrastructures, rSYBL uses Salsa[12] for the OpenStack private cloud, and Flexiant Cloud Orchestrator (FCO) for the case of the Flexiant public cloud, which differ in the protocols used and information required for the control.

For controlling the service on multi-clouds, the M2M DaaS stakeholder describes SYBL elasticity requirements, which are interpreted and enforced by rSYBL:

- *M2MDaaS*–STRATEGY CASE `avgBufferSize` < 5: minimize (`cost`)
- *LocalProcessingUnit*–CONSTRAINT `avgBufferSize`<50
- *EventProcessingUnit*–STRATEGY CASE `responseTime` < 40ms AND `throughput` < 20ops/s: `scalein`, CONSTRAINT `responseTime`<50ms

Moreover, the stakeholder adds a relationship that connects the two parts of the service deployed on multiple clouds, as shown in Listing 7.1, saying that we expect that whenever a huge amount of data is accumulated in the `LocalProcessingUnit` we would have huge amount of requests and vice-versa for the case of small amounts of data. This kind of information can be obtained from existing service analytics tools (e.g., Moldovan et al. [90]).

Listing 7.1: Relationship description

```
<Relationship type="Load" id="LoadRelationship">
    <source>LocalProcessingUnit</source>
    <target>LoadBalancerUnit</target>
    <metricSource>bufferSize</metricSource>
    <metricTarget>requests</metricTarget>
</Relationship>
```

[11] http://activemq.apache.org/

[12] http://github.com/tuwiendsg/SALSA

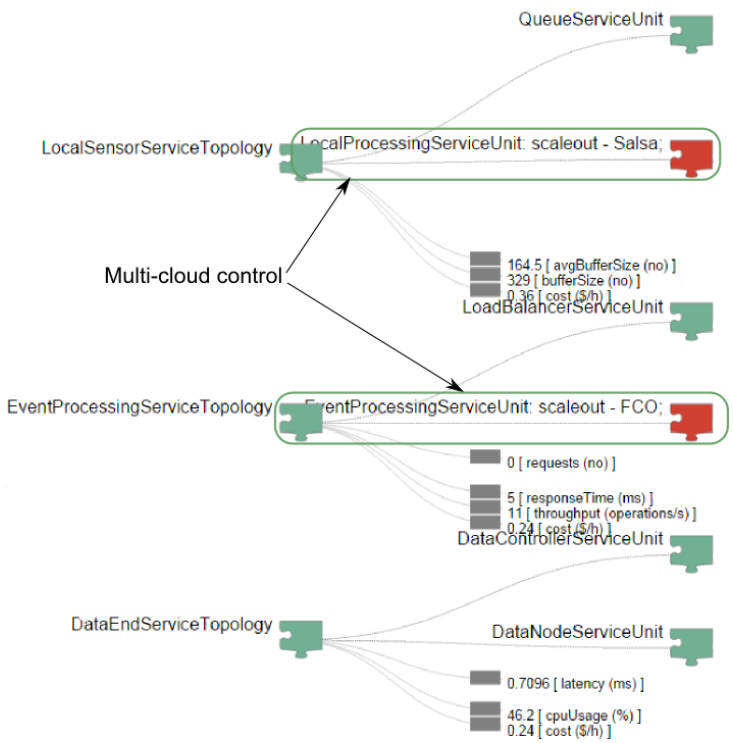


Figure 7.4: Multi-cloud control snapshot

Using the above information (e.g., M2M DaaS structure, associated primitives, requirements and relationships among service parts, and monitoring information from MELA[13]) that is represented in our runtime dependency graph, rSYBL generates and enforces when needed elasticity control plans containing sequences of elasticity capabilities for ensuring the fulfillment of elasticity requirements. The workload for our experiment consists of addition/removal of gateways with sensors, each gateway containing 3 to 15 sensors. Figure 7.4 shows the monitored M2M DaaS, with the metrics monitored for each service topologies and service units that belong to the service. Using the mechanism described in Algorithm 5, rSYBL decides that even though the constraint targeting `responseTime` is currently fulfilled, a new instance of `LocalProcessingUnit`, would increase the load, thus decrease the response time. For this, a compensation elasticity capability is added to the control plan: scale out `EventProcessingUnit` on Flexiant using FCO. Figure 7.5 shows the estimated cost associated to the private cloud, and cost associated to the public cloud, given that the buffer size has the shown peaks, due to the periodic sending of data to the public cloud. An increase of cost is associated with an increase of resources used by the M2MDaaS, and we can see that both on the public and private clouds the load intensity is followed. This way, the public cloud resources are allocated in advance to the load peak, for ensuring better elasticity and smaller cost on the M2M DaaS

[13]http://github.com/tuwiendsg/MELA

deployed on the public cloud, without having lags in provisioning sufficient resources.

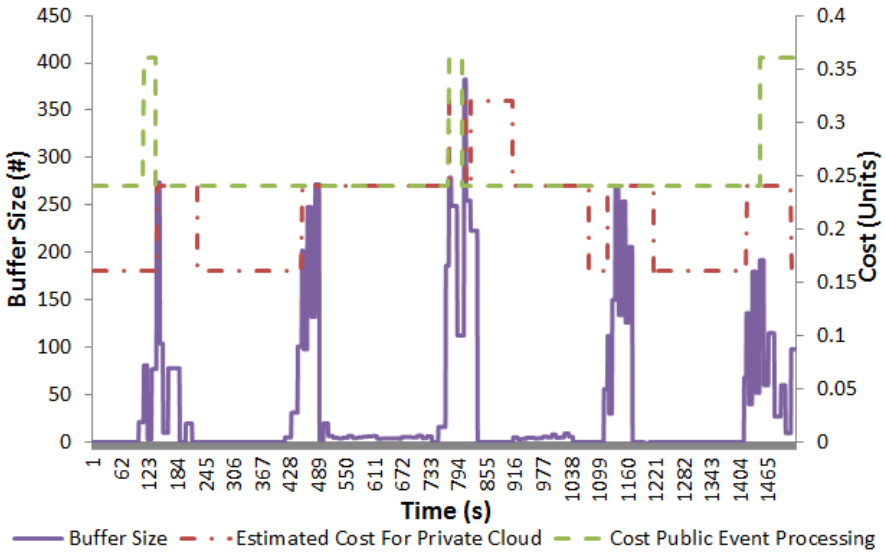


Figure 7.5: Multi-cloud executed M2M DaaS cost in time

Figure 7.6 shows the different amounts of time needed for enforcing elasticity capabilities on service units deployed on the two clouds. In this context, it is worth mentioning that Salsa deploys all artifacts needed on demand, while for Flexiant we use machines with needed software pre-installed. However, in both cases configurations are made on-demand, and the majority of the time is spent on creating and configuring the virtual machines. Therefore, we can affirm that in the case of Flexiant public cloud, the expected time for elasticity capabilities is much more reliable, with low standard deviation. In our experiment that lasted for 4 hours, in which each elasticity capabilities were enforced more than five times, the scaling out on Flexiant had a standard deviation of 0, while the scale in elasticity capability had a bigger deviation in average time, 2.82, due to the decomissioning (i.e., leaving the cluster) action that depends on the state of decomissioned unit.

We have shown that rSYBL is able to control an M2M service deployed on two different clouds, considering the end-to-end service perspective. For M2M service stakeholders, this is a step towards achieving better end-to-end service elasticity control, this way ensuring control for services possibly composed of multiple `LocalProcessingUnits` deployed over multiple mini-clouds.

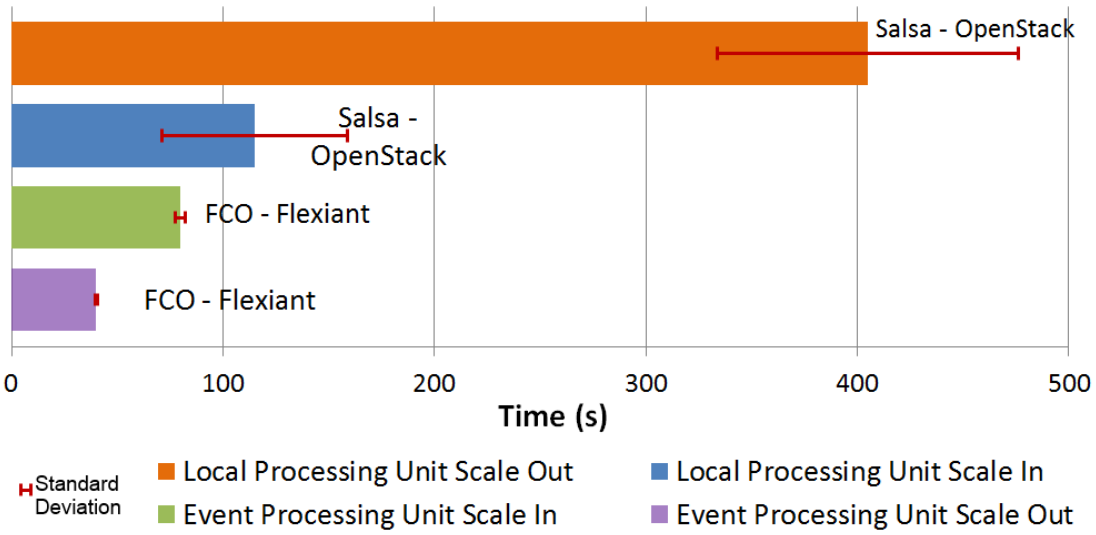


Figure 7.6: Multi-cloud control sensitivity

CHAPTER 8

Evaluating Cloud Service Elasticity Behavior

This chapter presents a novel methodology and a framework for estimating cloud service elasticity behavior. To estimate the cloud service behavior, we collect information concerning service structure, deployment, service runtime, control processes, and cloud infrastructure. Based on this information, clustering techniques are utilized to identify cloud service elasticity behavior, in time, and for different parts of the service. Knowledge about such behavior is utilized within a cloud service elasticity controller to substantially improve the selection and execution of elasticity control processes. These elasticity behavior estimations are successfully being used by our elasticity controller, in order to improve runtime decision quality.

8.1 Overview

With the wide adoption of cloud computing across multiple business domains, stakeholders seek to improve the efficiency of their complex cloud services, while also if possible to reduce costs, by acquiring on-demand virtualized infrastructure and, at the same time, benefiting from a pay-as-you-go price model offered by cloud providers. The key technique to achieve these goals is *elasticity* [1, 35] – the ability of cloud services to acquire and release resources on-demand, in response to runtime fluctuating workloads. From the customer perspective, resource elasticity can minimize task execution time, without exceeding a given budget. From the cloud provider perspective, elasticity provisioning contributes to maximizing their financial gain while keeping their customers satisfied and reducing administrative costs. However, automatic elasticity provisioning is not a trivial task.

To date, the user utilizes elasticity controllers, offered as a service, by either cloud

providers (e.g., Amazon Auto Scaling[1]) or third-party vendors (e.g., Rightscale[2]), to scale his/her distributed cloud services. A common approach, employed by many elasticity controllers [2] [128], is to monitor the cloud service and (de-)provision virtual instances for the service when metric thresholds are violated. This approach may be sufficient for simple cloud services, but for large-scale distributed cloud services with complex inter-dependencies among components, we need a deeper understanding of their elasticity behavior in order to select and enforce suitable elasticity control processes. For this reason, existing work [124] [128] has identified a number of elasticity control processes to improve the performance and quality of cloud services, while additionally attempting to minimize cost. However, a crucial question still remains unanswered: *which elasticity control processes are the most appropriate for a cloud service in a particular situation at runtime*? Moreover, can both cloud customers and providers benefit from insightful information such as *how the addition of a new instance to a cloud service will affect the throughput of the overall deployment and individually each part of the cloud service*? Thus, cloud service elasticity behavior knowledge under various control types and workloads is of paramount importance to elasticity controllers for improving their runtime decision making.

To this end, a wide range of approaches relying on service profiling or learning from historic information were proposed [110] [134]. However, these approaches limit their decisions to evaluating only low-level VM metrics (e.g., CPU and memory usage) and do not support elasticity decisions considering complex cloud service behavior at multiple levels (e.g., a specific part of the service or the entire service). Additionally, current approaches only evaluate resource utilization, without considering elasticity as a multi-dimensional property composed of cost, quality, and resource elasticity. Finally, existing approaches do not consider the outcome of a control process on the overall service behavior, where often enforcing a control process on the wrong part of the cloud service can lead to side effects, such as increasing the cost or decreasing performance of the overall service.

In this chapter, we focus on addressing the previous limitations by introducing a methodology for estimating cloud service elasticity behavior, and a corresponding framework named ADVISE (*evAluating clouD serVIce elaSticity bEhavior*). Our behavior estimation technique introduces a clustering-based process which considers heterogeneous information for computing expected elasticity behavior, in time, for various service parts. To estimate cloud service elasticity behavior, ADVISE utilizes different types of information, such as service structure, deployment strategies, and underlying infrastructure dynamics, when applying different workload and elasticity control processes. ADVISE analyses historical cloud service behavior, at various levels of abstraction, and produces estimations for elasticity control processes evaluated by the elasticity controller, *in time*, and for all cloud service parts, not only for the one targeted by the elasticity control process.

For validating our techniques, we integrate ADVISE in rSYBL [24] elasticity controller.

[1] `http://aws.amazon.com/autoscaling`

[2] `http://www.rightscale.com`

rSYBL is based on SYBL elasticity requirements specification language [25], which allows service providers to describe invariants and expected service behavior. rSYBL interprets requirements specified in SYBL, and based on these requirements it provides multi-grain elasticity control for complex cloud services. To evaluate ADVISE effectiveness, experiments were conducted on two cloud platforms with a testbed comprised of three cloud services originating from different service domains. Results show that ADVISE is able to determine the expected elasticity behavior, in time, with a low error rate (i.e., average standard deviation 0.46 over all considered elasticity control processes). Therefore, ADVISE can be integrated by cloud providers alongside their elasticity controllers to improve the decision quality, or used by service providers to evaluate and understand how various elasticity control processes impact their offered services. ADVISE and all the other tools used in this article (i.e., rSYBL, JCatascopia [119], and MELA [89]) are available as open source tools, thus enabling various stakeholders to apply them, or if needed to extend them, for obtaining elasticity in their respective domains.

8.2 Cloud Service Structural and Runtime Information

8.2.1 Cloud Service Information

For understanding the behavior of cloud services, we must gather multiple types of information, including application-specific behavior for different service parts and the various virtual resources used, and their characteristics.

To represent complex services (e.g., the case-study application in Section 4.2) in the context of behavior estimation, we extend the conceptual cloud service representation model, as proposed in Section 4.1, with a rich set of information types for determining cloud elasticity behavior. Figure 8.1 depicts the extensions made (white background) to include *elasticity control processes*, *service part behaviors* and *service parts*. Overall, our information model contains: (i) *Structural Information*, describing the architectural structure of the cloud service; (ii) *Infrastructure System Information*, describing runtime information regarding resources allocated for the cloud service by the underlying cloud platform; and (iii) *Elasticity Information*, capturing elasticity *metrics*, *requirements*, and *capabilities*.

Elasticity information is composed of elasticity metrics (e.g., average response time, cost, active connections), elasticity requirements (e.g., minimize response time when cost is small enough), and elasticity capabilities (e.g., add new resources), each of them being associated to different SPs or infrastructure resources. *Elasticity Capabilities* are grouped together as *Elasticity Control Processes* ($ECPs$), as described in the next subsection, and inflict specific elasticity behaviors upon enforcement on different SPs, which we model as *Service Part Behaviors*. We model SP behaviors, since controllers must determine the effect of enforcing an ECP at different levels. For instance, in the service previously described, before allocating a new *DataNodeServiceUnit* instance, the effect, in time, at the *DataEndServiceTopology* (e.g., latency evolution for the entire cluster), and at the entire cloud service level (e.g., number of violated requirements while enforcing the ECP)

should be determined.

Conceptually, a *Service Part Behavior*, denoted as $Behavior_{SP_i}$, in a defined period of time $[start, end]$, contains all the metrics, $M_a^{SP_i}$, being monitored for SP_i. Therefore, the behavior of a cloud service, denoted as $Behavior_{CloudService}$, over a period of time is defined as the set of all cloud service SP behaviors:

$$M_a^{SP_i}[start, end] = \{M_a(t_j) | SP_i \in ServiceParts, j = \overline{start, end}\} \quad (8.1)$$

$$Behavior_{SP_i}[start, end] = \{M_a^{SP_i}[start, end] | M^a \in Metrics(SP_i)\} \quad (8.2)$$

$$Behavior_{CloudService}[start, end] = \{Behavior_{SP_i}[start, end] | SP_i \in ServiceParts(CloudService)\} \quad (8.3)$$

The above information is captured and managed at runtime through an *Elasticity Dependency Graph*, $EDG = \{(V, E) | V \in SP \cup InfrastructureInfo, E \in Relationships\}$, which has as nodes instances of concepts from the model in Figure 8.1 (e.g., `Virtual Machine`, `Elasticity Metric`), and relationships (e.g., `Elasticity Relationship`, `Inheritance`, `Composition`) as edges. The elasticity dependency graph is continuously updated with: (i) *pre-deployment* information, such as service topology descriptions (e.g., TOSCA [98]) or profiling information; and (ii) *runtime* information, such as metric values from monitoring tools or allocated resources from provider APIs.

8.2.2 Elasticity Capabilities and Control Processes

Elasticity capabilities (ECs) are the set of actions associated with a cloud service, whose invocation affect the cloud service behavior. Such capabilities can be exposed by: (i)

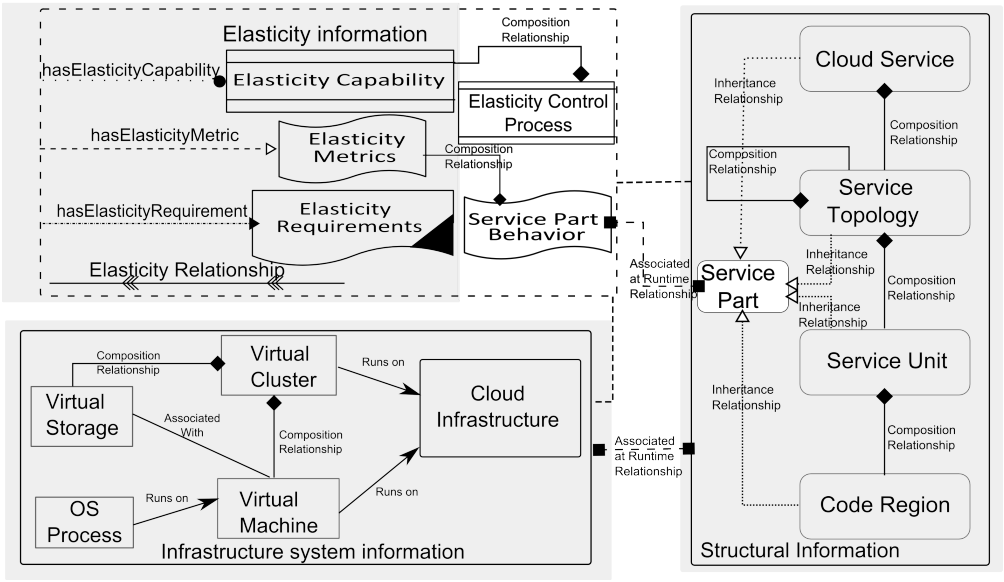


Figure 8.1: Cloud service information for estimating elasticity behavior

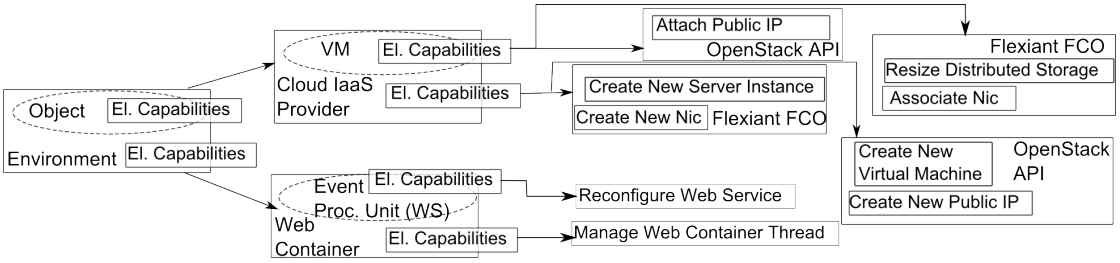


Figure 8.2: Elasticity capabilities exposed by different elastic objects

different SPs (e.g., change refresh rate for a SP); (ii) cloud providers (e.g., create new virtual resources); and (iii) resources which are supplied by cloud providers (e.g., change virtual resource characteristics). An EC can be considered as the abstract representation of API calls, which differ amongst providers and cloud services. Figure 8.2 depicts the different subsets of ECs provided for the *Event Processing Service Unit* (i.e., a web service hosted in a web container) when deployed on two different cloud platforms (e.g., Flexiant[3] and Openstack[4]), as well as the ECs exposed by the cloud service and its containers (e.g., Apache Tomcat). In each of the two cloud platforms, the cloud service must run on a specific container, and all these capabilities, when enforced by an elasticity controller, will affect various cloud service parts (e.g., in the M2M DaaS, elasticity capabilities of *Event Processing Service Unit* might affect the performance of the *Data End Service Topology*).

Elasticity Control Processes (ECP) are processes composed of elasticity capabilities, which can be abstracted into higher level capabilities having predictable effects on the cloud service. $ECPs$ can be in their simplest forms, sequential elasticity capabilities, while the more complex $ECPs$ are similar to business processes (e.g., enforcement plans from TOSCA described in BPMN). We model these $ECPs$ as graphs, $ECP = (V = \{EC\}, E = \{CF, DF\})$, where the vertices are elasticity capabilities, while edges are flow dependencies among elasticity capabilities. There are two types of flow dependencies: (i) Control Flow dependencies CF, which direct the execution of the process considering the initial state, and (ii) Data Flow dependencies DF, which carry data to be used by the next EC.

An ECP causes a change in the elasticity dependency graph and in the virtual infrastructures (e.g., a change in the properties of a single VM or tier). For example, in the case of a distributed database backend which is composed of multiple nodes, a scale out ECP, with certain parameters, can be applied for both a Cassandra and an HBase database, with the following ECs: (i) add a new node; (ii) configure node properties; and (iii) subscribe node to the cluster.

[3] `http://www.flexiant.com`
[4] `https://www.openstack.org`

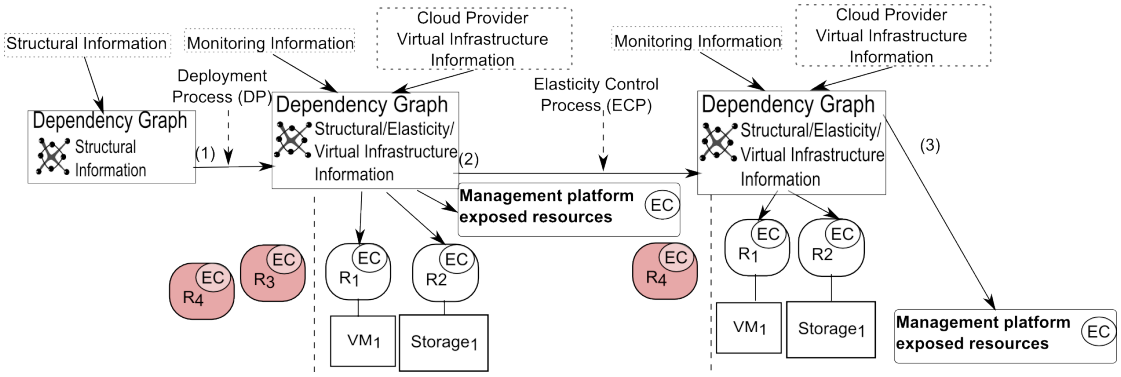


Figure 8.3: Elastic cloud service evolution

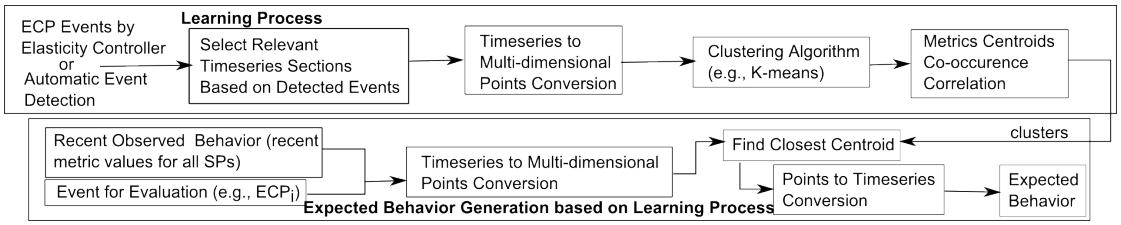


Figure 8.4: Modeling cloud service behavior process

8.2.3 Cloud Service Elasticity

To estimate the effects of $ECPs$ on SPs, we rely on the elasticity dependency graph which captures all the variables that contribute to cloud service elasticity behavior evolution. Figure 8.3 depicts on the left-hand side the cloud service at pre-deployment time, where automatic elasticity controllers know about it only from structural information provided by different sources (e.g., TOSCA description). After enforcing a Deployment Process (e.g., create VM, create network interface and connect to VPN), the elasticity dependency graph will be updated with infrastructure-related information obtained from the cloud provider, and elasticity information, obtained from monitoring services showing metric evolution for different SPs.

Infrastructure resources, as mentioned previously, have associated elasticity capabilities (EC in Figure 8.3), that describe the change(s) to be enforced and the mechanisms for triggering them (e.g., API call assigned to the EC). In addition, a cloud platform exposes ECs in order to create new resources or instantiate new services (e.g., increase memory is an EC exposed by a VM, while create new VM is an EC exposed by the cloud platform). In this context, for being able to discover the effects that an ECP produces in time, for each SP, taking into account correlations between metrics, we use the elasticity dependency graph. We analyze this information to determine the effect of an ECP for all SPs, regardless on whether the ECP is application specific, or it does not have any apparent direct link to other SPs.

8.3 Evaluating Cloud Service Elasticity Behavior

Existing behavior learning solutions [134] [110] learn discrete metric models, without correlating metrics with the multiple variables affecting cloud service behavior. In contrast to these solutions, we provide behavior learning based on different SPs and their relation to multiple $ECPs$, which may or may not be directly linked, estimating the effect of an ECP, *in time*, considering correlations among several metrics and SPs. Figure 8.4 depicts the SP behavior *Learning Process* which is continuously executed, refining the previously gathered knowledge base.

8.3.1 Obtaining Necessary Information

For evaluating cloud service elasticity behavior, we populate the dependency graph, described in Section 8.2, with all the necessary information (i.e., service parts, elasticity relationships, infrastructure system information). First, we acquire pre-deployment information to understand the cloud service and its execution environment, such as: (i) *structural information*, regarding the topology of the cloud service, and (ii) *cloud infrastructure information*. The first, described in Section 8.2, is generally known by the service provider, and contains the SPs of the service and relationships which appear among them. The latter describes virtual resources available in the current cloud infrastructure and their capabilities (e.g., a virtual machine of `type x` exposes the capability of memory ballooning). Afterwards, runtime information regarding the service behavior, is collected via monitoring tools (e.g., using MELA [89]), and associated with structural service units or topologies.

Monitoring data is either collected at runtime while a controller is enforcing different $ECPs$, or through a profiling step, where both rational and incorrect $ECPs$ are enforced. In both cases, issues may arise, such as application failure or virtual resource failure, leading to incomplete monitoring data. Therefore, we acknowledge two issue types: (i) *recurring issues*, which characterize a control step (e.g., a capability) and must be considered; and (ii) *random issues* which do not affect the behavior of a service. The first issue type must be reflected in the estimations, since they characterize the behavior of the service (e.g., while enforcing ECP_i SP_j cannot be monitored for X seconds). However, the second issue type can be ignored. For this, the following clustering methodology considers both, characterizing recurring behaviors (e.g., missing measurements each time a $type_x$ reconfiguration is enforced) and filtering outliers (e.g. random issues).

8.3.2 Learning Process

Figure 8.5 depicts the overall elasticity behavior clustering process via selected metrics observations, with the three main steps: (a) input data processing, (b) clustering process, and (c) behavioral clusters update.

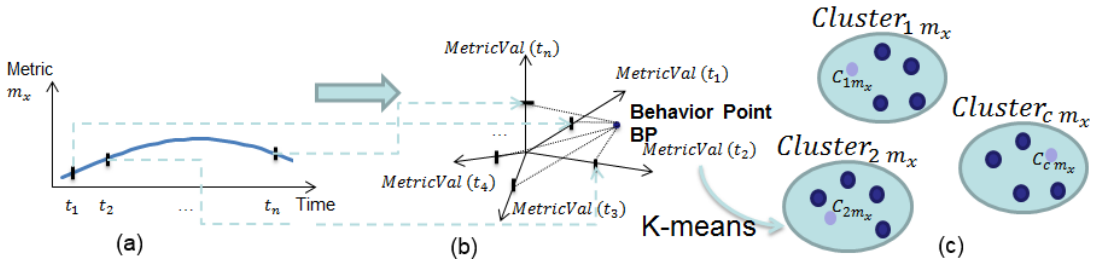


Figure 8.5: Clustering process

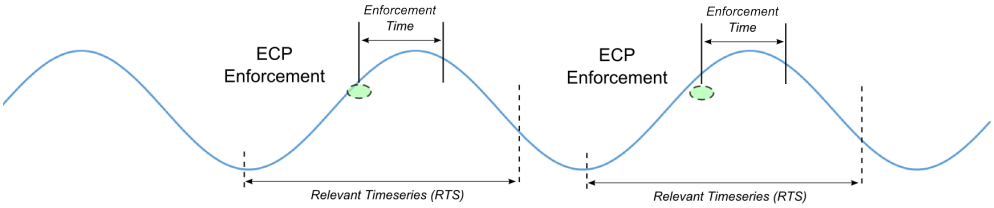


Figure 8.6: Relevant timeseries selection

Processing input data

The learning process receives as input each metric's evolution, in time, $M_a^{SP_i}[start, current]$ (see Equation 8.3) starting from the beginning of a service's lifecycle. To evaluate the expected metric evolution in response to enforcing a specific ECP, we select for each monitored metric, of each service part, a *Relevant Timeseries Section* (RTS) (see Figure 8.6), in order to compare it with previously encountered $M_a^{SP_i}[start, current]$. The RTS size strongly depends on the average time required to enforce an ECP (see Section 8.5.2). Consequently, a metric RTS is a sub-sequence of $M_a^{SP_i}$, ranging from an interval before and after ECP enforcement:

$$RTS_{M_a}^{SP_i} = M_a^{SP_i}[x - \frac{\delta + ECP_{time}}{2}, x + \frac{\delta + ECP_{time}}{2}], \tag{8.4}$$
$$[ECP_{startTime}, ECP_{endTime}] \subset [x - \frac{\delta + ECP_{time}}{2}, x + \frac{\delta + ECP_{time}}{2}],$$

where x is the ECP index and δ is the length of the period we aim to evaluate.

As part of the input pre-processing phase, we represent $\delta + ECP_{time}$ (Figure 8.5a) as multi-dimensional points (Figure 8.5b and Equation 8.5) in the n-dimensional Euclidian space, where the value for dimension $t(j)$ is the timestamp j of the current RTS.

$$BP_a^{SP_i}[j] = RTS_{M_a}^{SP_i}[t(j)], j = 0, ..., n, BP : M^{SP} \mapsto R^n, n = \delta + ECP_{time} \tag{8.5}$$

Clustering process

To detect the expected behavior of an ECP enforcement result, we construct behavioral point clusters $Cluster_{SP_i}$, for all SPs and each ECP as defined in Equation 8.6. We do

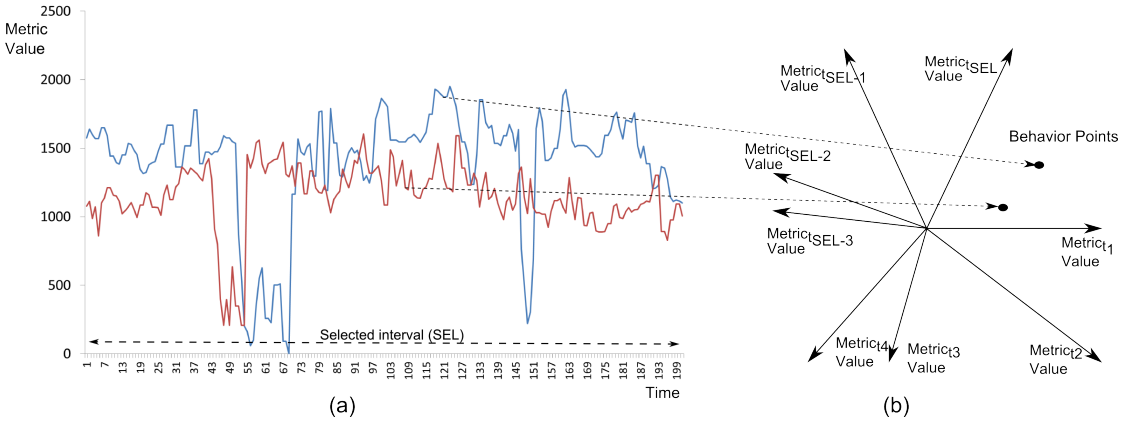


Figure 8.7: Relevant timeseries sections to points

not limit our approach to only considering $ECPs$ available for the current SP_i since, as previously mentioned, enforcing an ECP to a specific SP may affect the behavior of another SP or the overall cloud service. Our objective function is to find the multi-dimensional behavior point $C(\Theta^*)$, which minimizes the distance among points belonging to the same cluster $Cluster_k$ (Equation 8.7). Since our focus is not to evaluate the quality of different clustering algorithms, we use the K-means algorithm, which is inexpensive, following the practice where the number of clusters is $K = \sqrt{N/2}$, N being the number of objects. However, as shown in Section 8.5, even with a simple K-means algorithm, our approach outputs the expected elasticity behavior with a low estimation error rate.

$$dist(BP_a^x, BP_a^y) = \sqrt{\sum_i (BP_a^x[i] - BP_a^y[i])^2} \tag{8.6}$$

$$\Theta^* = \arg\min \sum_{k=0}^{K} \sum_{i=0}^{N} \Theta_{i,k} dist(Cluster_k, BP_i), \quad \theta_{i,k} = \begin{cases} 1 & BP_i \in Cluster_k \\ 0 & BP_i \notin Cluster_k \end{cases} \tag{8.7}$$

Behavioral clusters update

For the update process, we start from the already existing clusters, and we search for new behavior points given by new ECP enforcements. We select relevant timeseries RTS for each SP in response to newly enforced $ECPs$, whenever new data is available, according to the process presented in Section 8.3.2. We represent these as behavior points BP, and add each of them to the closest clusters. The cluster update process then consists of moving BPs among the clusters until convergence, which is a lightweight process compared to running entire clustering algorithm. The overhead of updating the clusters is proportional with the number of selected $RTSs$ and the change in cloud service behavior. Even with $ECPs$ enforced very often, the cluster updating process is still insignificant due to the fact that the RTS dimensions are quite small compared to the overall monitoring data. However, it must be noted that repeated $ECPs$ in short

intervals reflect a weakness in the controller, which does not manage to stabilize the system as we can see in section 8.5.

8.3.3 Determining the Expected Elasticity Behavior

Elasticity Behavior Correlation

After obtaining $\|\delta + ECP_{time}\|$-dimensional point clusters, we construct for each SP_i a co-occurrence matrix $CM_{SP_i}[C_x^{m_i}, C_y^{mj}]$, where C_x is the centroid for $Cluster_x$, and the value of CM is the probability of clusters from metrics m_i and m_j to appear together (e.g., increase in data reliability is usually correlated with increase in cost).Considering this, when determining expected behavior points, we have a better estimation that is also based on correlation among metrics(e.g., when in M2M DaaS the *EventProcessingServiceUnit* `throughput` is high, a `Scale IN` ECP will increase `response time`, while when low, the impact might be negligible). An item in the CM represents a ratio between the number of times the behavior points C_x and C_y were encountered together, and the total number of behavior points. This matrix is continuously updated when behavior points move from one cluster to another, or when new $ECPs$ are enforced, thus, increasing the knowledge base.

Expected Behavior Point Determination

In the *Expected Behavior Generation based on Learning Process* step in Figure 8.4, we select the latest metric values for each SP_i, $M_a^{SP_i}[current - \delta, current]$, and the ECP_ξ which the controller is considering for enforcement, or for which the user would like to know the effects. We find the $ExpectedBehavior$ (see Equation 8.8) which consists of a tuple of cluster centroids from the clusters constructed during the *Learning Process* that are the closest to the current metrics behavior for the part of the cloud service we are focusing on, and which have appeared together throughout the execution of the cloud service. The result of this step, for each metric of SP_i, is a list of expected values from the enforcement of ECP_ξ (e.g., all of the expected metric values for the case the elasticity controller would like to perform a scale out ECP).

$$ExpectedBehavior[SP_i, Behavior_{SP_i}[current - \delta, current], ECP_\xi] = \{C_{i_{a^1}}^{M_{a^1}}, ..., C_{i_{a^m}}^{M_{a^m}} | M_{a^m} \in Metrics(SP_i)\} \quad (8.8)$$

8.3.4 Parameterizing and qualifying the learning process

The quality of the expected elasticity behavior estimation depends on several, highly configurable, variables, which are either: (i) the output of a pre-profiling process; or (ii) empirically determined, based on good practices or on observing the behavior of the estimation process. Such variables include: (i) variable K, denoting the number of expected clusters; (ii) the variable $cutoff$, denoting the acceptable clustering convergence error; and (iii) the monitoring information completeness. For K, as defined above, we follow the rule of thumb proposed by Mardia et al. [82], stating that the number of

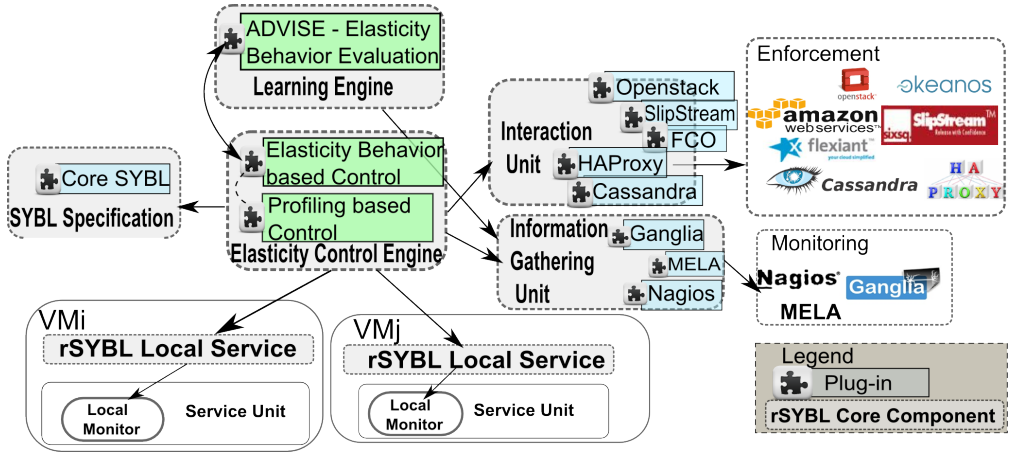


Figure 8.8: ADVISE integration into rSYBL

clusters in a set of objects is $\sqrt{N/2}$, where N is the total number of objects. The offset is empirically determined, considering the quality of the results and the time needed for computing an estimation, as shown in Section 8.5.2.

8.4 Controlling Elasticity with Elasticity Behavior Estimation

We have implemented our elasticity estimation techniques in a framework named ADVISE. As described in Section 8.2, ADVISE collects the following heterogeneous types of information, for populating the dependency graph: (i) cloud service *structural information*, from TOSCA service descriptions; (ii) *infrastructure* and *application performance information*, from both JCatascopia [119] and MELA [89] monitoring systems; and (iii) *elasticity information*, regarding *ECPs* from the rSYBL [24] elasticity controller.

We extend rSYBL to integrate ADVISE to support both understanding the behavior of the different service parts and enforcing control processes in accordance. Figure 8.8 depicts the rSYBL framework integrated with ADVISE for estimating elasticity behavior and considering it when generating *ECP* plans. The rSYBL *Elasticity Control Engine*, now features two planning mechanisms: (i) *profiling-based control*, enabled when ADVISE-based estimations are not yet accurate enough (e.g. average standard deviation is low), and (ii) *elasticity behavior-based control*, which considers runtime elasticity behavior estimations for all service parts in order to evaluate the needed elasticity control process(es). Moreover, ADVISE also exposes statistics on the time required for an *ECP* to be enforced, thus being able to refine the enforcement *cool-off* period on the controller side as presented in Section 8.5.2.

Profiling-based control requires as input, per *ECP*, the expected effects obtained through manual or automated profiling. The effects that are available for this case are forecasts for expected metric values after finishing enforcing the respective *ECP*, and

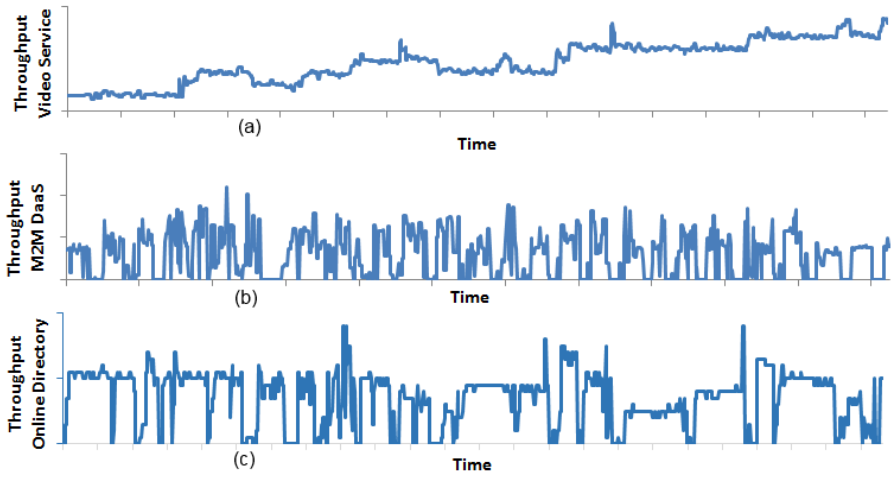


Figure 8.9: Workload applied on the three services

do not consider relationships among various SPs. The $ECPs$ are enforced through the rSYBL Interaction Unit, which interacts with both cloud provider-specific APIs and with application specific control mechanisms.

For generating control plans, we interpret the effect, in time, of $ECPs$ in relation to expected behavior without enforcing the respective control process. In this sense, *elasticity behavior-based control* estimates current metric evolution using a polynomial fitting approach. We obtain the elasticity requirements expected to be violated when we do not enforce ECP_ξ (Equation 8.9) by computing the integral of violated requirements over estimated polynomial metric evolutions. Equation 8.10 shows the computation method for violated elasticity requirements for the case of enforcing ECP_ξ, denoted $ViolatedReq(ECP_\xi)$, using ADVISE estimations. We compare the two estimations, and select the one with the least violated requirements.

$$ViolatedReq(ECP_\xi) = \int_{current}^{current+\lambda} ViolReq(P(SP_i, Metric_j(x)))dx \quad (8.9)$$

$$ViolatedReq(ECP_\xi) = \int_{current}^{current+\lambda} ViolReq(ADVISE(SP_i, Metric_j(x)))dx \quad (8.10)$$

We design the controller such that when the current behavior cannot be accurately estimated, we are able to rollback to profiling-based control. In this way, the enforced $ECPs$ are not restricted solely to information obtained via the pre-deployment phase, instead the ADVISE behavior estimation is continuously refined improving the knowledge base and learning new behavior points representative for each ECP. In other words, when the closest estimated centroids are farther than a distance `dist`, empirically defined, the current decision making algorithm rolls back to the initial decision-making algorithm, as described in [24].

8.5 Experiments

In this section, we provide an evaluation of the ADVISE framework[5], focusing on the clustering-based behavior estimation process to determine the effectiveness of our approach as ADVISE can be used in both service profiling/pre-deployment or during runtime, for various service types, whenever monitoring information and enforced $ECPs$ are available. As described in Section 8.2, ADVISE collects the following heterogeneous types of information, from plugins which we have developed, to populate the elasticity dependency graph: (i) cloud service *structural information*, from TOSCA service descriptions; (ii) *infrastructure* and *application performance information*, from both JCatascopia [119] and MELA [89] monitoring systems; and (iii) *elasticity information*, regarding $ECPs$ from the rSYBL [24] elasticity controller.

Our evaluation is divided into two phases: (i) ADVISE framework evaluation; and (ii) ADVISE-enabled rSYBL evaluation. To evaluate the functionality of the ADVISE framework, we established a testbed on the Flexiant public cloud comprised of three cloud services originating from different service domains featuring distinct structural and behavior requirements. On the selected cloud services, we first, enforce $ECPs$ exposed by their respected SPs randomly, and then study, at runtime, their behavior at multiple levels of the cloud service. It must be noted, that we did not configure rSYBL as a rational controller, since we are interested in estimating the elasticity behavior for all SPs as a result of enforcing both good and bad elasticity control decisions. For the second phase, we established a testbed on an OpenStack private cloud, with rSYBL elasticity control for a cloud service. We evaluate how ADVISE affects rSYBL elasticity control on various workloads.

8.5.1 Experimental Cloud Services

The **first cloud service** is a **three-tier web application** providing **video streaming services** to online users, comprised of: (i) an *HAProxy Load Balancer* that distributes client requests across multiple application servers; (ii) An *Application Server Tier*, where each application server is an Apache Tomcat server exposing the video streaming web service; and (iii) A Cassandra *NoSQL Distributed Data Storage Backend* from where the necessary video content is retrieved. The database backend initially holds 2GB of data while at the end of the experiment the size approaches 6GB. To stress this cloud service we generate client requests under a fixed request rate, though the load is not stable and depends on the type of the requests (e.g. download video) and the size of the requested video, as shown in the workload pattern in Figure8.9a.

The **second cloud service** in our evaluation is a **Machine-to-Machine (M2M) DaaS** which processes information originating from several different types of remote data sensors (e.g., temperature, atmospheric pressure, or pollution). Specifically, the M2M DaaS is comprised of an *Event Processing Service Topology* and a *Data End Service Topology*. Each service topology consists of two service units, one with a processing

[5] code, detailed descriptions and more charts at `http://tuwiendsg.github.io/ADVISE`

Cloud Service	ECP ID	Action Sequence
Video Service	ECP_1	*Scale In Application Server Tier*: (i) stop the video streaming service, (ii) remove instance from HAProxy, (iii) restart HAProxy, (iv) stop JCatascopia Monitoring Agent, (v) delete VM
	ECP_2	*Scale Out Application Server Tier*: (i) create new network interface, (ii) instantiate new VM, (ii) deploy and configure video streaming service, (iv) deploy and start JCatascopia Monitoring Agent, (v) add VM IP to HAProxy, (vi) restart HAProxy
	ECP_3	*Scale In Distributed Video Storage Backend*: (i) select VM to remove, (ii) decommission instance data to other nodes, (iii) stop JCatascopia Monitoring Agent, (iv) delete VM
	ECP_4	*Scale Out Distributed Video Storage Backend*: (i) create new network interface, (ii) instantiate new VM, (iii) deploy and configure Cassandra (e.g., assign token to node), (iv) deploy and start JCatascopia Monitoring Agent, (v) start Cassandra
M2M DaaS	ECP_5	*Scale In Event Processing Service Unit*: (i) remove service from HAProxy, (ii) restart HAProxy, (iii)remove recursively VM
	ECP_6	*Scale Out Event Processing Service Unit*: (i) create new network interface, (ii) create new VM, (iii) add service IP to HAProxy
	ECP_7	*Scale In Data Node Service Unit*: (i) decommission node (copy data from VM to be removed), (ii)remove recursively VM
	ECP_8	*Scale Out Data Node Service Unit*: (i) create new network interface, (ii) create VM, (iii) set ports, (iv) assign token to node, (v) set cluster controller, (vi) start Cassandra
Online Directory	ECP_9	*Scale In Distributed Document Store*: (i) select Couchbase-server to remove, (ii) decommission node from Couchbase cluster, (iii) rebalance cluster data, (iv) remove VM
	ECP_{10}	*Scale Out Distributed Document Store*: (i) create new network interface, (ii) instantiate VM, (ii) configure interfaces, ports, and Couchbase-server (iv) start Couchbase-server, (v) join Couchbase cluster, (vi) rebalance cluster

Table 8.1: Elasticity control processes available for the cloud services

goal, and the other acting as the balancer/controller. To stress this cloud service we generate random sensor event information (see Figure 8.9b) that is processed by the *Event Processing Service Topology*, and stored/retrieved from the *Data End Service Topology*.

The **third cloud service** showcased is a **two-tier OLTP service** deployed as an **online business directory** hosting 7503 local business listings and their products[6]. The topology of this service is comprised of a *Document Store Controller*, under a

[6]Our dataset is synthetic, created from real data and workload patterns from `www.finditcyprus.com`

Cloud Service	SP Name	Metrics
Video Service	Application Server Tier	cost, busy thread number, request throughput
	Distributed Video Storage Backend	cost, CPU usage, memory usage, query latency
M2M DaaS	Cloud Service	cost per client per hour
	Event Processing Service Topology	cost, response time, throughput, number of clients
	Data End Service Topology	cost, latency, CPU usage
Online Directory	Document Store Controller	cost, request rate, active sessions, error rate, CPU usage, network I/O
	Distributed Store Node	cost, throughput, cache miss rate, disk I/O, memory usage, CPU usage, query response time

Table 8.2: Elasticity metrics per cloud service for different service parts

public domain, distributing client requests (i.e., create new listing, get directions to *Restaurant X*) to a *Document Store Nodes*, forming a *Distributed Document Store* that is a Couchbase database backend. Specifically, the database backend is a distributed, shared-nothing NoSQL (JSON-like) document store, optimized for interactive web applications, incorporating in its core application logic allowing developers to prepare and expose to their users queries as light-weight map/reduce functions (i.e., top-k *breweries* in town *Nicosia* etc.). We stress this service by generating client requests under a variable read-heavy request rate, mimicking a real online directory's behavior, as depicted in Figure 8.9c (writes occur only when adding a new listing or updating an existing one and constitute less than 10% of the load). Tables 8.1 and 8.2 list the $ECPs$ associated to each SP and the monitoring metrics analyzed for the three cloud services respectively.

8.5.2 Cloud Service Elasticity Behavior Evaluation

ECP Temporal Effect

ADVISE computes, as shown in Table 8.3, the *average time* required for an ECP to be completed, and returns also a standard deviation that gives the degree of confidence with regard to this estimation. This application-specific information is of high importance and affects the decision-making process of the elasticity controller since it is an indicator of the *grace period* that it should await until effects of the resizing actions are noticeable. Thus, it can define the time granularity at which resizing actions should be taken into consideration. For example, we observe that the process of reconfiguring and removing an instance from the video service's storage backend requires an average time interval of 160 seconds that is mainly due the time required to receive and store data from other

	ECP	Standard Deviation	Average ECP Time (s)
Video Service	ECP1	0.06	90
	ECP2	0.12	25
	ECP3	0.13	160
	ECP4	0.11	30
M2M Service	ECP5	0.34	45
	ECP6	0.16	20
	ECP7	0.11	70
	ECP8	0.14	20
Online Directory	ECP9	0.29	110
	ECP10	0.12	25

Table 8.3: Elasticity control processes time statistics

nodes of the ring. If decisions are taken in smaller intervals, the effects of the previous action will not be part of the current decision process and may cause cascading *ping-pong* effects where a Scale In ECP followed by a slight increase in metric utilization (in the *grace* period) causes a false Scale Out ECP.

Online Video Streaming Service - Elasticity Behavior Estimation

Figure 8.10 depicts both the *observed* and the *estimated* behavior for the Video Service Application Server Tier when $ECPs$ of type ECP_1 (*remove application server*) are enforced. At first, we observe that the average `request throughput` per application server is decreasing. This is due to two possible cases: (i) the video storage backend is under-provisioned and cannot satisfy the current number of requests which, in turn, results in requests being queued; (ii) there is a sudden drop in client requests which indicates that the application servers are not utilized efficiently. For an elasticity controller driven by simple "if-then-else" policies for application-specific metrics (e.g. request throughput) there is no apparent way in determining the case in hand and it will act upon metric violations without considering if an ECP will indeed improve QoS or cost. From Figure 8.10, we observe that after the Scale In ECP occurs, the average `request throughput` and `busy thread number` rises which denotes that this behavior corresponds to the second case where resources are now efficiently utilized. ADVISE revealed an insightful correlation between two metrics to consider in the decision process.

Similarly, in Figure 8.11 we depict both the *observed* and the *estimated* behavior for the Distributed Video Storage Backend when a Scale Out ECP is enforced (add Cassandra Node to ring) due to high `CPU utilization`. We observe that after the Scale Out ECP is enforced, the actual `CPU utilization` decreases to a normal value as also indicated by the estimation.

Analyzing the estimations made for this service (i.e., Figure 8.10 - 8.11), we conclude that the estimations provided by ADVISE successfully follow the actual behavior pattern and that, as time intervenes, the curves tend to converge.

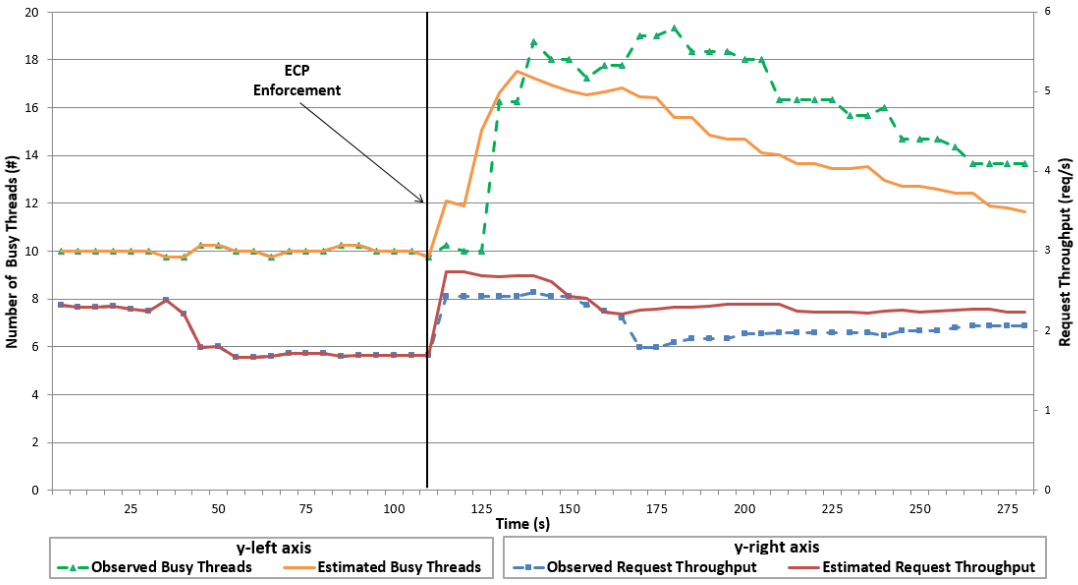


Figure 8.10: Effect of ECP_1 on the application server tier

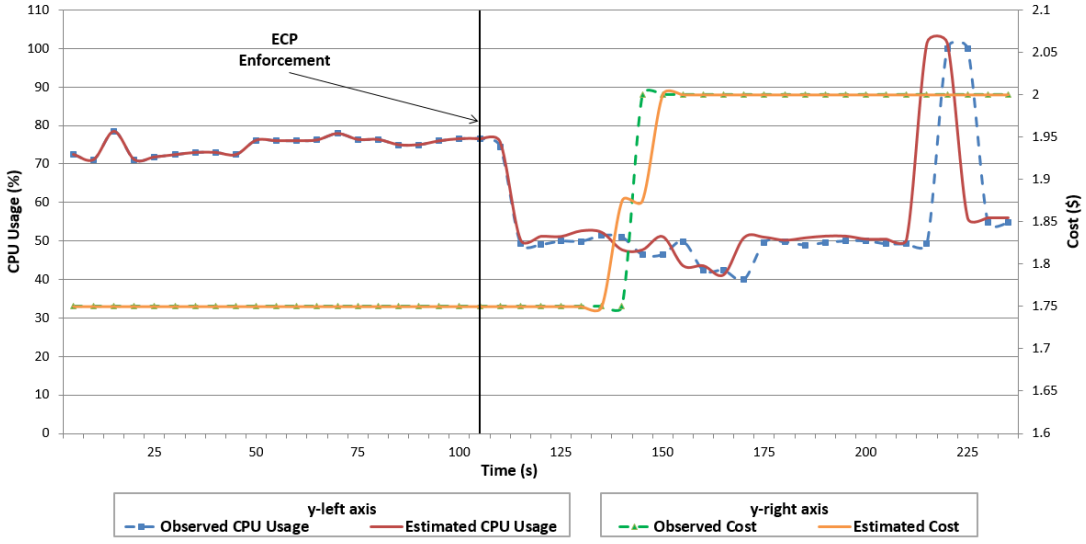


Figure 8.11: Effect of ECP_4 on the entire video streaming service

M2M DaaS - Elasticity Behavior Estimation

Figure 8.12 displays how an ECP targeting a service unit affects the entire cloud service. The `Cost/Client/h` is a complex metric (see Table 8.2) which depicts how profitable is the service deployment in comparison to the current number of users. Although `Cost/Client/h` is not accurately estimated, due to the high fluctuation in number of clients, our approach approximates how the cloud service would behave in terms of

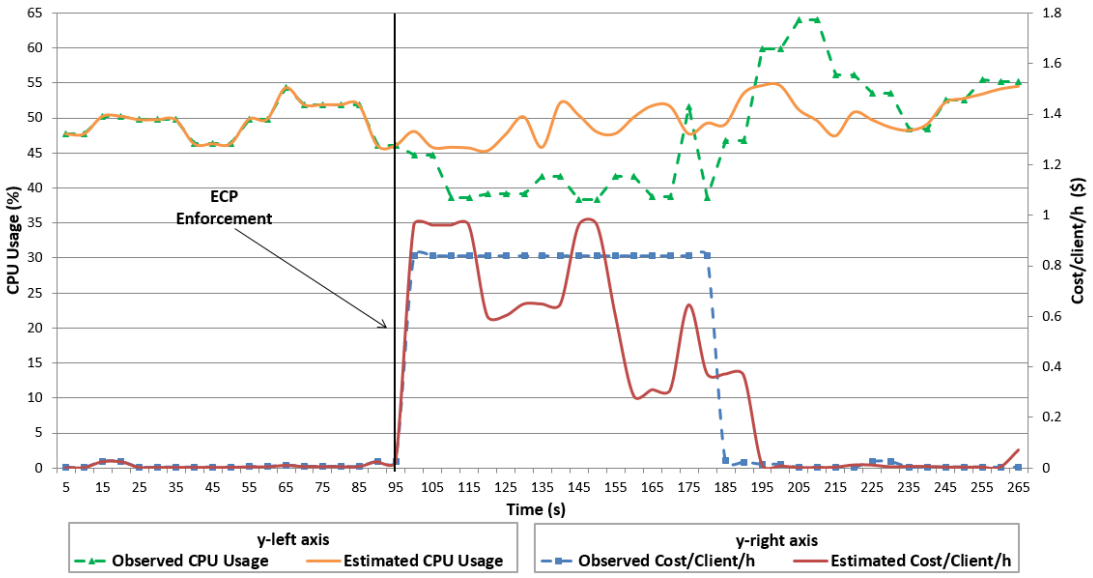


Figure 8.12: Effect of ECP_7 on M2M DaaS

expected metric fluctuations. This information is important for elasticity controllers to improve their decisions when enforcing *ECPs* by knowing how the `Cost/Client/h` for the entire cloud service would be affected. Although `CPU usage` is not estimated perfectly, since it is a highly oscillating metric, and it depends on the `CPU usage` at each service unit level, knowing the baseline of this metric can also help in deciding whether this *ECP* is appropriate (e.g., for some applications `CPU usage` above 90% for a period of time might be inadmissible). Figure 8.13 shows estimations of behavior for the Event Processing Service Topology, when a Scale Out *ECP* occurs on the Event Processing Service Unit. Although the `throughput` is accurately estimated with a slight lag, `response time` is estimated with a slightly larger error due to the fact that a down peak is not estimated, as not being part of the usual behavior for the current SP.

ADVISE can estimate the effect of an *ECP* of a *SP*, on a different *SP*, even if apparently unrelated and therefore provide, multi-grain elasticity behavior evaluation. Figure 8.14 depicts an estimation on how the Data Controller Service Unit is impacted by the data transferred at the enforcement of ECP_8. In this case, the controller `CPU usage` initially, as one would aspect, decreases since the new node will offload other nodes, however, effort is required in transferring data to the new node which rises utilization due to the fact that reconfigurations are also necessary on the controller, following a slight decrease and then stabilization. Therefore, even in circumstances of random workload, ADVISE can give useful insights on how different *SPs* behave when enforcing *ECPs* exposed by other *SPs* which, again, elasticity controllers have no knowledge of.

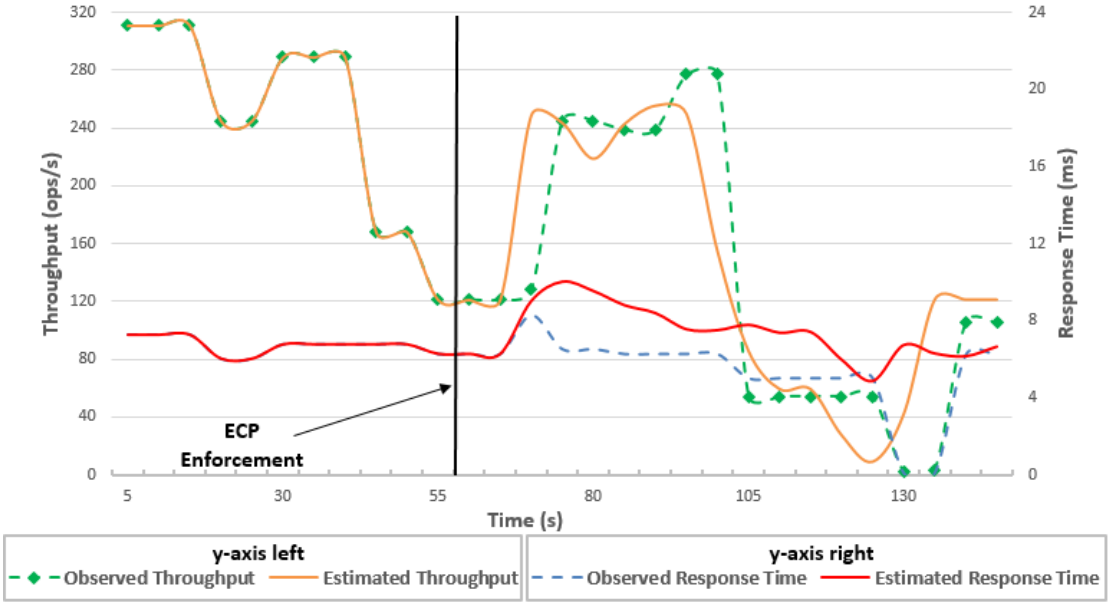


Figure 8.13: Effect of ECP_6 on the event processing service topology

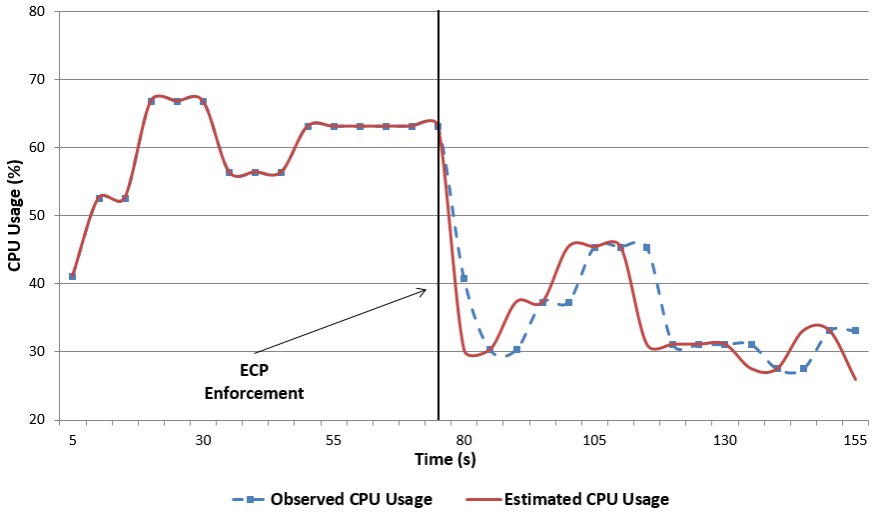


Figure 8.14: Effect of ECP_8 on the Data Controller Service Unit

Online Directory - Elasticity Behavior Estimation

Figure 8.15 depicts the *observed* and *estimated* `throughput` and `CPU usage` measured at the Document Store Controller after a Scale Out ECP is enforced. The Document Store Controller is a document store node itself, however, it features additional functionality: it *supervises (meta-)data migration* when the cluster is re-balanced as in the case of a Scale In/Out ECP enforcement. While other nodes continue to accept client requests

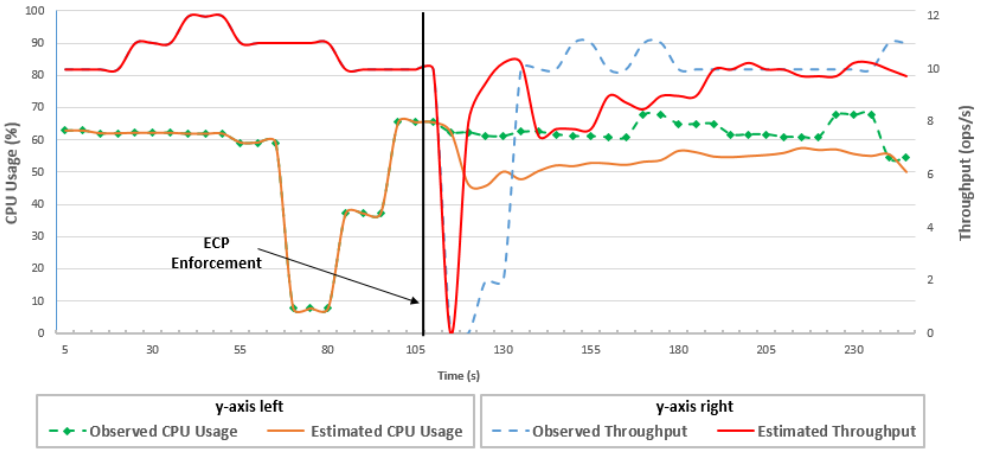


Figure 8.15: Effect of ECP_{10} on the Document Store Controller

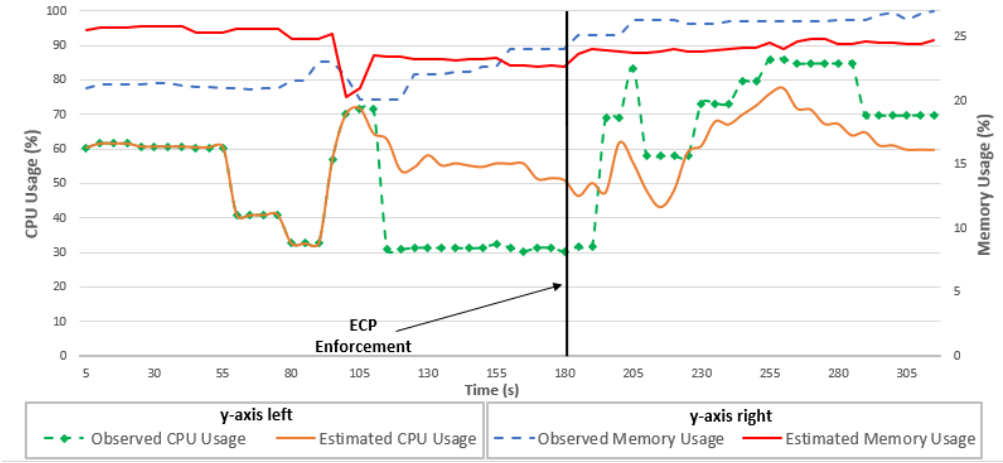


Figure 8.16: Effect of ECP_9 on the Document Store Node

when an ECP is enforced, the Document Store Controller prioritizes the supervising of data movement and thus, ceases to process client requests. This is evident in Figure 8.15, where we observe that when the cluster is rebalanced, `throughput` drops to zero while `CPU usage` does not decrease, as one would expect, until after rebalancing is complete. In turn, Figure 8.16 depicts the effects of a Scale In ECP on one of the document store nodes. ADVISE identifies the increase in `memory utilization` as the node receives part of the load from the decommissioned node, while the estimation for `CPU utilization` follows the observed oscillating trend.

On the other hand, Figure 8.17 depicts both `CPU usage` and `cost` of a document store node before and after a Scale Out ECP. We observe that before the ECP enforcement the ADVISE estimation follows the observed values, however, after the ECP enforcement, the provided estimation slightly deviates from the observed `CPU`

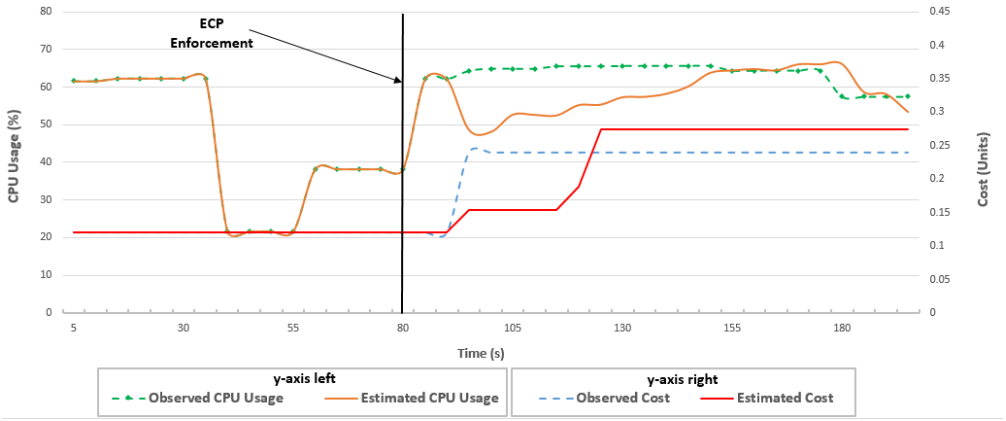


Figure 8.17: Effect of ECP_{10} on the Data Node

`utilization` before converging again. The reason behind this slight deviation is due to, an out of the ordinary, large data movement, not evident in most Scale Out $ECPs$, where the whole dataset must be replicated since the Scale Out ECP occurs immediately after a series of multiple Scale In ECPs which left the cluster with only two instances.

Quality of Results

ADVISE is able to estimate, in time, the elasticity behavior of different SPs by considering the correlations amongst metrics and the $ECPs$ which are enforced. To evaluate the quality of our results, we have considered the fact that existing tools do not produce continuous-time estimations. Thus, we evaluate ADVISE by computing the variance Var and standard deviation $StdDev$ (Equation 8.11), over 100 estimations as the result differs little afterwise.

$$Var_{metric_i} = \frac{\sum \frac{\sum_{i=[0,rts_{size}]} (estMetric_i - obsMetric_i)^2}{rts_{size}}}{nbEstimations - 1}$$
$$StdDev_{metric_i} = \sqrt{Var_{metric_i}} \tag{8.11}$$

Table 8.4 presents the accuracy of our results. When comparing the **three services**, the Video Service achieves a higher accuracy (smaller standard deviation), since the imposed workload is considerably stable. Focusing on the M2M DaaS estimation accuracy, we observe that it depends on the granularity at which the estimation is calculated, and on the ECP. Moreover, the standard deviation depends on the metrics monitored for the different parts of the cloud service. For instance, in the case of the M2M Service, the `number of clients` metric can be hardly predicted, since we have sensors sending error or alarm-related information. This is evident for the Event Processing Service Topology, where the maximum variance for the `number of clients` is 4.9.

Overall, even in random cloud service load situations, the ADVISE framework analyses and provides estimations for elasticity controllers, allowing them to improve the quality of

Cloud Service	Observed Cloud Service Part	Elasticity Control Process	Average Standard Deviation	Maximum Variance	Minimum Variance
Video Service	Video Service	ECP_3	0.23	0.09	0.03
		ECP_4	0.61	0.99	0.23
	Distributed Video Storage Backend	ECP_3	0.28	0.14	0.034
		ECP_4	0.2	0.042	0.04
	Application Server	ECP_1	0.43	0.4	0.06
		ECP_2	0.31	0.47	0.01
M2M Service	Cloud Service	ECP_5	0.9	6.65	0.24
	Data End Service Topology	ECP_5	0.23	0.35	7.44E-05
	Event Processing Service Topology	ECP_7	1.1	4.9	0.046
		ECP_8	0.76	2.46	0.027
	Data Controller Service Unit	ECP_6	0.12	0.25	0
		ECP_8	0.22	0.41	0
	Data Node Service Unit	ECP_5	0.572	0.68	0.32
		ECP_6	0.573	1.4	0.07
	Event Processing Service Unit	ECP_7	1.08	3.59	0.11
		ECP_8	0.77	1.9	0.14
Online Directory	Document Store Node	ECP_9	0.19	0.05	0.29
	Document Store Node	ECP_{10}	0.14	0.005	0.18
	Document Store Controller	ECP_{10}	0.13	0.023	0.38

Table 8.4: ECPs effect estimation quality statistics

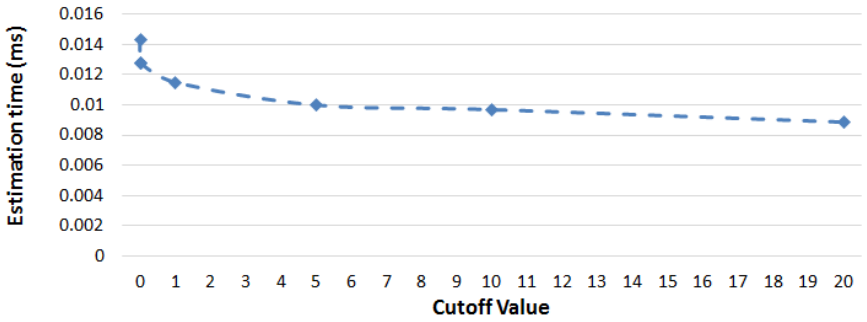


Figure 8.18: ECP_5 estimation time under different Cutoff values

control decisions, with regard to the evolution of monitored metrics at the different cloud service levels. Moreover, these estimations are delivered together with the confidence of the estimation, given by the distance from current behavior point to the estimated behavior point. Without this kind of estimation, elasticity controllers would need to use VM-level profiling information, while having to control complex cloud services. This

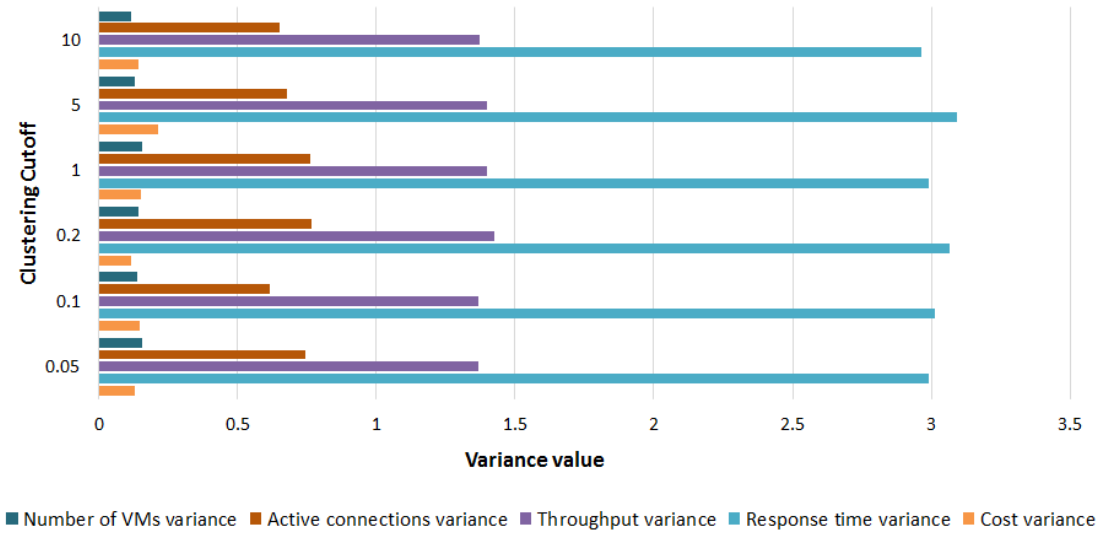


Figure 8.19: Estimation variance for ECP_5 under different Cutoff values

information, for each SP, is valuable for controlling elasticity of complex cloud services, which expose complex control mechanisms.

Sensitivity of results

For analyzing the sensitivity of our results, we evaluated how our empirically determined parameters (e.g. clustering offset) affect the variance of the estimation in regards to the observed behavior. Figure8.19 concludes that our results are very little affected by the choice of the offset.

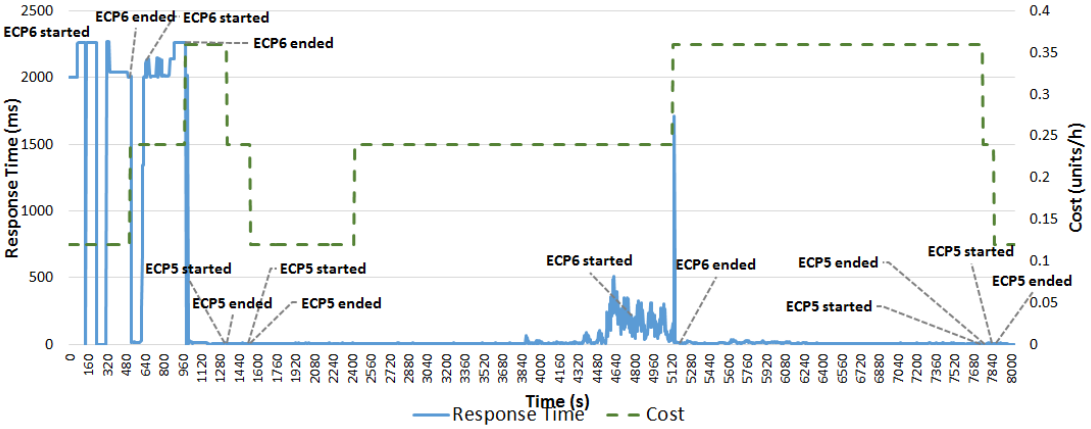
When analyzing the impact that the choice of the offset has over time (Figure 8.21b), we can see that very small offset values reflect in a considerable increase in the estimation time. This is why, the offset was chosen at `0.2`, as a tradeoff between estimation time and estimation quality.

8.5.3 rSYBL elasticity control enhanced with ADVISE estimations

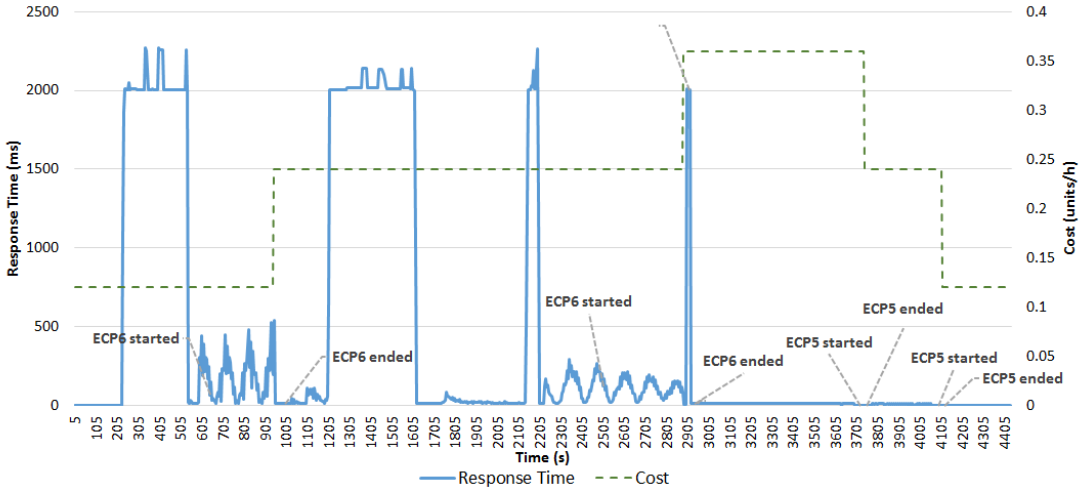
As described in Section 8.4, we integrate the ADVISE behavior estimation into the rSYBL elasticity controller, which controls elasticity at multiple levels of abstraction [24].

We use the M2M DaaS service with fixed *Data End Topology*, while controlling *Event Processing Topology* using ECP_5 (i.e., Scale In) and ECP_6 (i.e., Scale Out) elasticity control processes on an OpenStack private cloud, with the following SYBL [25] elasticity requirements:

- *EventProcessingTopology* – Co1:CONSTRAINT `responseTime` < 100 ms;



(a) rSYBL with ADVISE knowledge

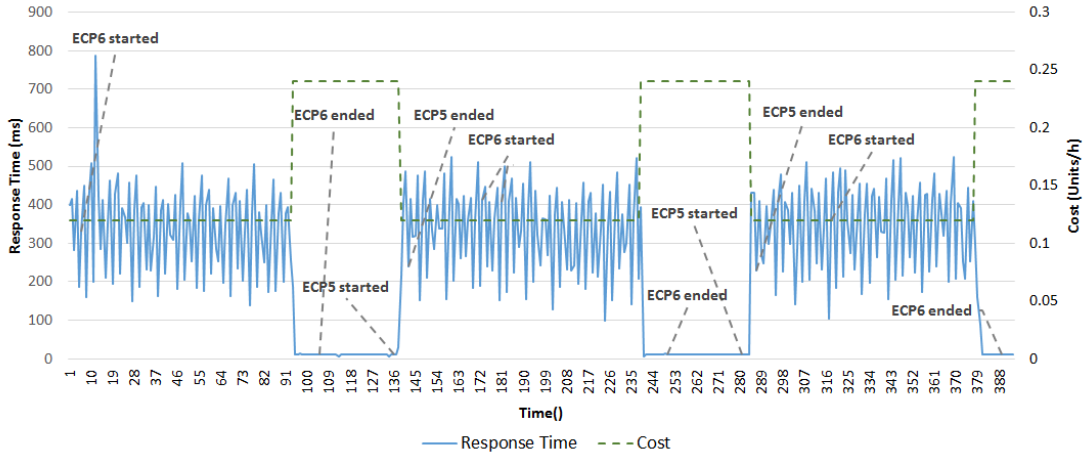


(b) rSYBL without ADVISE

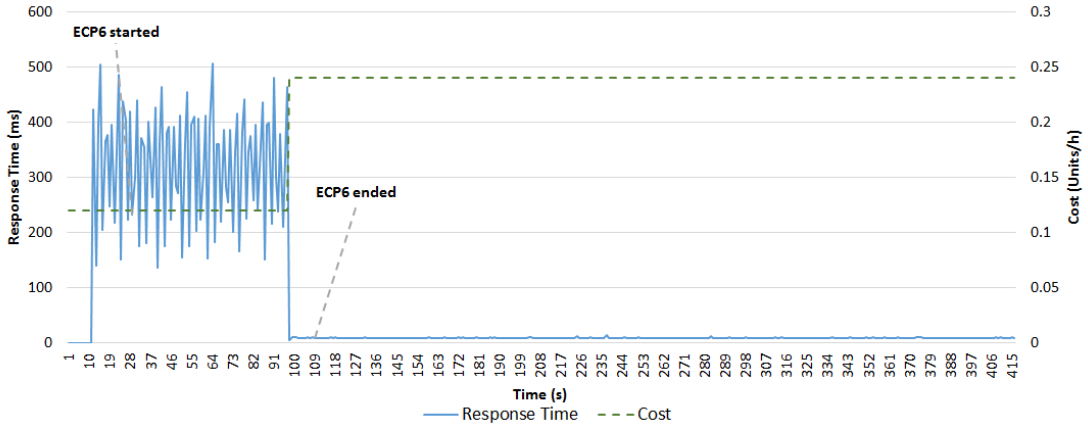
Figure 8.20: Event Processing Topology control

- *EventProcessingTopology* – St1:STRATEGY CASE `responseTime` < 12 AND `throughput`<100 : minimize(`cost`)

We compare the elasticity control performed by ADVISE-enabled rSYBL decision making and respectively the profiling-based decision making. We apply a stepwise workload in order to observe controller's behavior under different circumstances. Figure 8.20 shows at (a) the outcome of controlling the service with ADVISE-enabled rSYBL and at (b) the outcome of controlling the service considering profiling information, consisting of how much the metrics would be affected by enforcing an *ECP*. At first, we observe that ADVISE-based control provides the elasticity controller with a better elasticity behavior understanding, even in cases where metric values exceed their expected values (i.e. in this case due to a memory bottleneck). However, it accomplishes this with a bigger cost, as



(a) "Ping-pong" effect for steady workload with rSYBL



(b) No "ping-pong" when using ADVISE

Figure 8.21: Ping-pong effect

one would expect, since it is using more resources to fulfill the requirements (e.g., in this case ADVISE-enabled controller achieves a cost 27.5% higher). In contrast with this, the profiling based elasticity controller is not able to find appropriate $ECPs$ in unexpected situations, when the value observed exceeds its possibilities of controlling the metric, as known from profiling information (e.g., if the response time is expected to decrease with 200 ms whenever $ECP6$ is enforced, while current metric values are 2000 ms). Moreover, the ADVISE-based control considers the following interval when analyzing the expected behavior, not only the final metric values. Thus, whenever enforcing an ECP results in comparable requirements violation over the whole period as for the case of no ECP enforcement, the controller chooses to not take any action.

When applying a steady/fix workload, depending on the formulated requirements,

some controllers can reach continuous control oscillations. This was also the case for rSYBL with profiling information, as we can see from Figure 8.21 (a). Due to the continuous effects which are being used by rSYBL with ADVISE, on the same workload, and the same elasticity requirements, this "ping-pong" effect is avoided (see Figure 8.21 (b)), since the controller knows that the enforcement of $ECP5$ will imply the overall increase of the `response time`, after a time period.

Using ADVISE in elasticity control can also avoid situations where various control processes are enforced without understanding their outcome. For instance, enforcing Scale Out when response time satisfies a condition does not always result in response time decrease. Moreover, in some cases, where the service is not truly elastic, enforcing $ECPs$ considering expected discrete effects would only cause increase in costs. With ADVISE and SYBL, the control processes are enforced only when, considering current and previous behavior, the ECP would help fulfilling SYBL requirements (e.g., increase throughput, minimize response time).

CHAPTER 9

Elasticity Operations Management

This chapter analyzes the needs of service providers and the possible interactions in elasticity operations management that should be supported. We focus on interactions between service provider employees and elasticity controllers, and propose novel interaction protocols considering various organization roles and their concerns from the elasticity control point of view. The elasticity Operations Management Platform (eOMP) is presented, which supports seamless interactions among service provider employees and software controllers.

9.1 Overview

A service deployed in the cloud can make use of various resources and services offered by cloud providers, and can be very dynamic at run-time. The cloud is one of the most dynamic environments: providers can change their cost schemas, and the offered service characteristics, from one day to another. Although in this environment automated controllers are necessary, given this high dynamism, it might be necessary for service stakeholders to re-examine the desired behavior of a cloud service, and their interactions with other stakeholders (e.g., cloud providers, or data providers). For instance, whenever the load of the cloud service dramatically changes, the normal "safety requirements" (e.g., do not exceed a specific cost value) might not hold. For these situations, service providers have employed analysts to oversee the control process, and detect when such a case is encountered by a controller. A much better solution would be that the controller itself notifies the responsible person with the encountered situation (e.g., unexpected behavior or service health issues). Moreover, when multiple service provider employees manage service elasticity requirements, they might introduce requirements conflicts, which result in control instability, or in sub-optimal controller behavior. In such situations,

the employees responsible for the detected cases, once notified, should be able to easily interact with the controller to solve the detected issues. This type of "human-in-the-loop" based control not only improves the runtime elasticity customization capabilities, but also empowers service providers with more control over their services and automated elasticity controllers. Thus, for obtaining service elasticity at runtime, the service provider needs this kind of support in its operation phase (i.e., from service management lifecycle), in order to carry out the service operation processes in the cloud environment. Such control over incidents and over the provided service quality is of utter importance in dynamic environments, where there are a multitude of variables that can affect the service behavior, some of which being manageable only manually (e.g., renegotiating contracts, deciding on the quality of service to be provided to customers, and adapting to unexpected failures). Although currently several solutions provide elasticity control of cloud services (e.g., [81], [63]), service provider employees are not included in the elasticity control process.

For addressing the challenges above, in this chapter we propose adding *roles* (i.e., service provider employees) as first class entities in cloud service elasticity control loops. Based on the roles, we define necessary interaction protocols for managing service elasticity operation phase. We focus on interactions between roles and elasticity controllers, but we also support simple interaction among employees, for notifying each other of updates or for delegating responsibility for incidents. We extend SYBL [25], a language for expressing elasticity requirements, to support roles and role-based communication between stakeholders. Based on this, we develop an elasticity Operations Management Platform (eOMP) for cloud services, and we validate its usefulness showing various events encountered for a complex service. eOMP can adapt to various organization structures, and enables service provider employees to interact easily with the elasticity controller, for obtaining a more complex elasticity control. eOMPs supports managing unexpected situations, by facilitating the collaboration between elasticity controllers and service employees, identifying various types of events occurring during operation phase, and providing mechanisms for solving them.

9.2 Motivation

For managing cloud services, a service provider needs to prepare its employees for issues that can appear in cloud environments, and to adapt their internal processes to this kind of issues (e.g., service health issues, or change in cloud provider offerings). Moreover, given cloud environment's dynamism, it might be necessary to constantly analyze the service at runtime, to make sure that the service customers receive the expected quality of service. For this it may be necessary to use an automated elasticity controller, for rapidly reacting to environment changes. The elasticity controller can release from the employees responsibilities, but could change some responsibilities, for including the elasticity controller better in the service operation management process.

Figure 9.1 shows a complex environment ranging from IoT-specific resources (i.e., sensors, actuators) up to the cloud environment, where information gathered by the

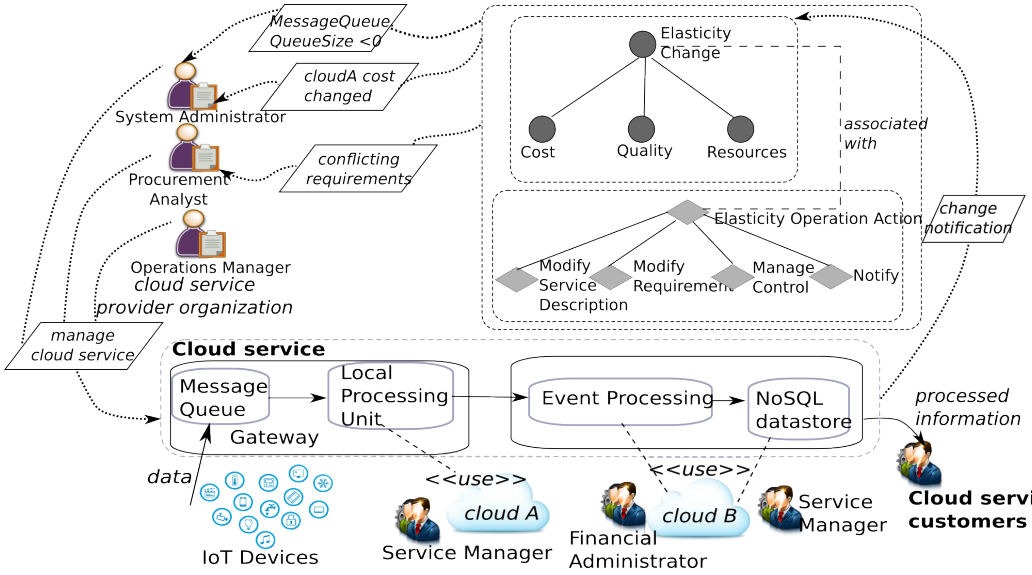


Figure 9.1: Motivating scenario

sensors is stored and processed. In cloud infrastructures, we consider a complex cloud service using various types of software (e.g., NoSQL databases, RabbitMQ) that processes the data coming from sensors and exposes it to various service customers (e.g., fire department, or building management application). In this chapter, we consider cloud service elasticity as being controlled at multiple levels of abstraction, following the cloud service model presented in [24], where the cloud service is composed of units (i.e., elementary parts of the service), which can be grouped into topologies (i.e., semantically connected units).

Focusing on the service provider, we can see that it needs to fulfil various types of requirements (e.g., data placement for IoT device users, or providing expected quality for service customers). For managing the services running in the cloud, the service provider needs its employees to follow clearly defined **r**oles. For improving the management efficiency, the service provider normally needs to use an automated elasticity controller (e.g., rSYBL [24]), for increasing the reaction speed and the control quality. While it is obvious that nowadays decision-making solutions should be as much software-based as possible, given the complexity of the setting, there are situations in which it is necessary for the decisions to be taken by real persons. Depending on the type of change that has appeared in the service (i.e., following elasticity dimensions: resources, cost, quality [36]), various roles can have different interests or responsibilities in the cloud service control. For instance, whenever the cost of running the platform in the cloud gets high, a *financial administrator* might analyze the overall evolution, and decide whether as a strategic decision it makes sense to use more virtual resources than initially estimated. For this, s/he could either invest more in the platform, or negotiate with *cloud providers* for better prices. Although most times service elasticity is achieved simply

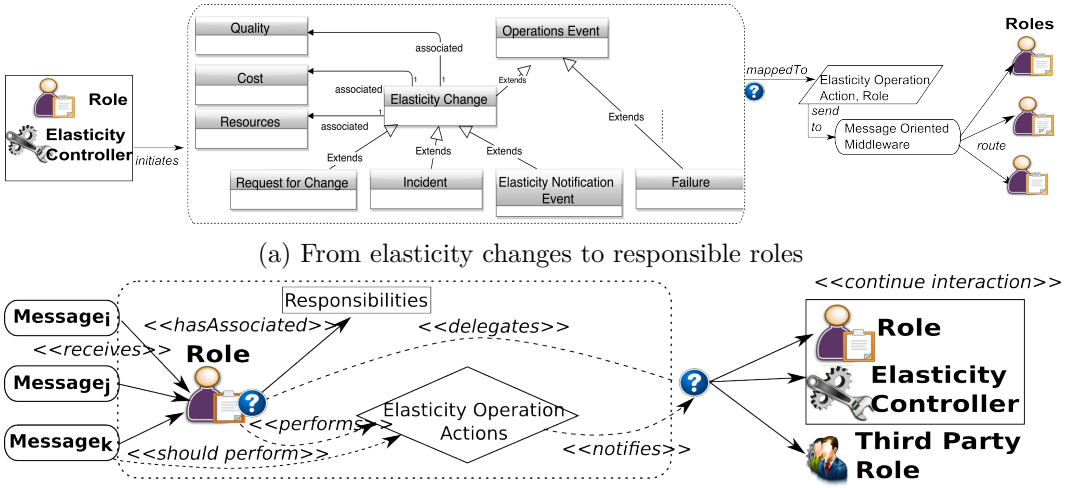
by allocating more/less virtual resources, similar effects might be achieved by changing configurations (e.g., changing the load balancing mechanism). In this case, employees in charge with *configuration management* should know the configuration being used. The service provider's goals can also evolve in time, due to varying number or types of users. In our scenario, when adding further data providers for the service, the control requirements need to be modified by the responsible employees, e.g., ensure better data transfer, even though the cost would go over what before was specified as the maximum admissible cost.

Therefore, although an automated solution is necessary for this case, the Level of Automation (LOA) [38] of the elasticity controller should not be of *Full Automation*, but the human (service provider employee) should be included in the control loop, for supporting the cases discussed above, in a *Supervisory Control* mode. Moreover, for achieving this kind of interaction between the elasticity controller and the different types of service provider employees, clear interaction protocols are needed. Since existing solutions mainly focus on full automation ([63], [81]), we propose using supervisory control for cloud services elasticity, and define interaction protocols for elasticity control. Thus, we introduce a platform easing service provider's interaction with the elasticity controller, at multiple levels of authority and for multiple elasticity concerns.

9.3 Analyzing Interactions in Elasticity Operations Management

9.3.1 Role interactions

As described in our motivation, the main focus of our work is supporting service providers for achieving better supervised elasticity control. Different service provider employees are normally in charge with different operations, thus being associated with various roles as part of the organization. We use in our analysis the term *roles* instead of stakeholders or employee types, since different persons/stakeholders/employee types can play various roles, possible multiple roles at a time. Moreover, while the roles present in a company are rarely dynamic, the stakeholders/employee types and persons in a company are both volatile and dynamic. Following our motivation scenario, the roles and elasticity controller need to collaborate in order to manage service operation during runtime. As shown in Figure 9.2a, roles should be notified by other roles or by the elasticity controller concerning the operation events that occur in relation to the elasticity of current service. From the operation events, we focus on any type of elasticity changes. We characterize elasticity changes according to the elasticity dimensions (i.e., resources, cost, and quality), and focus on three types of elasticity change events: (i) request for change (RFC), (ii) incident, and (iii) elasticity notification event. The request for change event can be initiated by either an elasticity controller or a role, requesting for changes in properties of the service. The elasticity capabilities (i.e., changes that can be enforced at runtime for modifying service behavior) can incur unexpected behavior in quality, cost or resources, which can indicate an issue in the service configuration or the deployed



(a) From elasticity changes to responsible roles

(b) From roles to elasticity operation actions

Figure 9.2: Role interaction flow

artifacts. Moreover, service providers are interested in failure events, in order to be able to learn from service behavior and environment changes that produce failures. The elasticity changes need to be mapped into elasticity operations actions and the roles that need to receive them, using knowledge that characterizes the service provider processes and is managed by the elasticity operations management platform. As shown in Figure 9.2b, a role can receive a multitude of messages from other roles or from elasticity controllers. Analyzing them, the role decides whether it can perform needed actions, or if it should delegate to other roles. After analyzing the interaction flow, the next section is focused on analyzing the roles and their responsibilities and interests in elasticity operations management.

9.3.2 Elasticity operations and roles

In the case of elastic cloud services the elasticity controllers play a big role in management, as opposed to ordinary service management roles [15]. Lower level authority roles (i.e., system administrator) have limited responsibilities, supervising and delegating work to the elasticity controller. We focus on designing interaction between elasticity controllers and several roles[1], excluding the roles whose functionalities are replaced by the elasticity controller(e.g., Performance/Capacity Analyst, Systems Operator, or Capacity Manager).

We identified the following operations management roles sensible to the cloud service elasticity:

- *Service Manager* - responsible for Service Support and Service Delivery actions taken in order to meet IT requirements (responsible for user satisfaction). This

[1] http://www.itsmcommunity.org/downloads/ITIL_Role_Descriptions.pdf

role is interested in using elasticity for achieving best performance and quality at best possible cost.

- *Incident Analyst* - provides support role to receive elasticity changes that cannot be automatically resolved. From elasticity control perspective, to this role should be reported any detected incidents for which the controller has no solutions. This role is interested in the possible incidents that are discovered, or even introduced, while performing elasticity control processes. While when the service is deployed in a static environment, this role mainly interacts with the system administrator, and operations manager, for the case when the service is deployed in a cloud environment, it can receive valuable information from the elasticity controller.
- *Problem Owner/Manager* - responsible of reviewing problem trends. In the case of cloud service whose elasticity is automatically controlled, this role should receive problems notifications from the elasticity controller. Moreover, the frequency of the notifications should differ with the gravity of the detected problems.
- *Test Manager* - ensures proper testing for changes released into production. Whenever un-predictable behavior is detected by the cloud service elasticity controller, the test manager is notified for checking if it is the desired behavior.
- *Configuration Librarian* - responsible for maintaining up-to-date records of configuration items. When the possible runtime modifications that the elasticity controller is allowed to perform include configuration changes, the configuration librarian should be notified concerning the changes. Depending on the change frequency, it may be required that it receives an aggregated list of configurations performed.
- *IT Financial Manager/ IT Financial Administrator* - monitors cost evolution and produces reports on IT assets and resources used by the service. When the analyzed service is deployed on a cloud environment, this role is interested in receiving the cost of hosting the service on possibly multiple cloud providers.
- *Operations Manager* - provide necessary services/resources in order to meet SLAs. When deployed in traditional servers (i.e., not cloud), the operations manager is in charge with managing all the roles that manage the infrastructure (e.g., Asset Administrator, Physical Site Engineer, Hardware Engineer). In the current case, when the service is deployed on the cloud and we use an elasticity controller, the mentioned roles are substituted by the elasticity controller, and the Operations Manager would need to interact alot with the elasticity controller.
- *Systems Administrator* - administers infrastructure (servers, hosts, networking devices). The system administrator can use the elasticity controller to automate most of his/her tasks.
- *Systems Operator* - performs operational processes ensuring that services meet operational targets. The system operator can use the elasticity controller to automate most of his/her tasks.
- *Procurement Analyst* - contact for vendor suppliers. Is in charge with finding the vendors for the needed services, negotiating the costs and signing contracts. In cloud computing, part of these responsabilities are delegated to the elasticity controller. However, some responsabilities that involve humans and cannot be replaced by

Modification type	**R**oles interested
Changes in cost due to scaling/changing the infrastructure/software services that are being used	IT Financial Manager, Service Manager
Changes in cost due to providers change in cost, without change in performance	Procurement Analysis, Service Manager, Operations Manager
Changes in quality due to providers change in quality, without change in cost	Service Manager, Operations Manager
Requirement on cost inflicts degradation of performance	Operations Manager
Configuration change due to change of the workload	Configuration Librarian, System Administrator, Operations Manager
Service part that is not healthy (erroneous, not behaving as expected)	Test Manager, Operations Manager, System Administrator, Incident Analyst
Changes in quality w/o change in workload	Operations Manager, Service Manager
Requirements that are not fulfilled/ are on danger of not being fulfilled by the cloud service elasticity controller	Operations Manager
Data compliance requirements changed by the data provider	Service Manager, Operations Manager

Table 9.1: Examples of elasticity modifications and roles interested

software (e.g., cloud providers do not offer re-negotiation APIs).

Table 9.1 shows the possible elasticity-driven behavior modifications that can be triggered by the elasticity controller, and the roles that might be interested in these types of modifications. Depending on the frequency of modifications, the roles are interested of receiving events more or less often, events being aggregated from a number of modifications, or containing a single modification. For instance, cost-related modifications need to be viewed by finance-related roles, like IT Financial Manager, Procurement Analysis, or Service Manager. Quality-related events (e.g., service part is not healthy, requirements not fulfilled) are of interest for Operations Manager and Service Manager.

9.4 Elasticity Operations Management Platform

We design our platform (Figure 9.3) following the flow described in the previous section, for supporting interactions between roles and elasticity controllers, and even third party roles. The controller sends messages through an *Embedded Queue*, for message locality reasons. From eOMP, the *Control Communication* component processes the received notifications. Here we provide a plugin-based mechanism for ensuring that eOMP can be adapted to different elasticity controllers, the only constraint being to map controller notifications to the eOMP model with operation events. The *Control Communication* component processes the operation events, and depending on their types, and depending on the

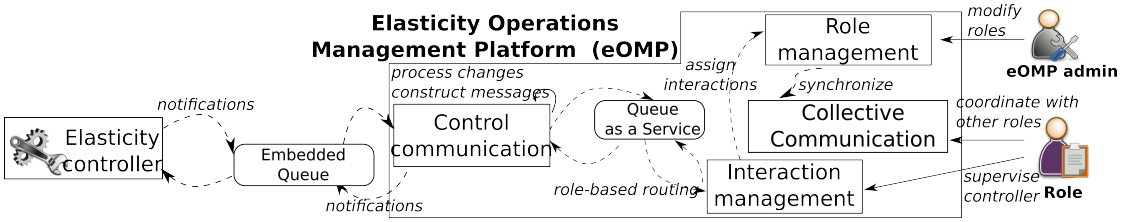


Figure 9.3: eOMP design

responsibilities of the roles in the *Role Management Component*, maps the controller messages to correct interactions with current roles, and adds them to the queue. For this, we use a *Queue as a Service*, since depending on the number of roles, and possibly in the future, organizations, that we want to support, the scale of the queue and routing complexity can increase. Queue interactions are consumed from the roles' side by the *Interaction processing* component, which directly interacts with the role (e.g., user interface, command line, API-driven communication). For communication between roles, we use a component for collaborative interactions in human-based computing systems (e.g., SmartCOM[2]).

We designed the platform in such a way that the elasticity controller is agnostic to IT service management processes, or role types. This is beneficial since it enables service providers to choose other controllers, or, for the case of very small company (i.e., one employee), to use the controller without the rest of our platform. The roles and responsibilities can be modified by the eOMP administrator, e.g., by using roles from a different standard.

9.4.1 Entities of the Interaction

For defining the interactions that may occur for the cloud service elasticity control, we focus on the participants in interactions. We use *organization* to describe an association with a functional structure (e.g., service provider organization). We focus firstly on modeling interactions between the service provider organization and the elasticity controller, and secondly on modeling limited interactions among organizations.

Roles (Eq. 9.2) are entities that are associated with responsibilities R and authority levels Aut, and that can be assigned to different employees at various points in the organization lifetime. An employee can have multiple roles at a given time (e.g., both systems administrator and systems operator). Each organization is defined by a set of role types, which change rarely throughout organization lifetime (e.g., when adopting CMMI[3], depending on the maturity level, roles used change). *Third party roles* are roles from partner organizations, which are known to the current organization, and are accessible only through an internal role for various actions (e.g., notifications, or contract

[2] https://github.com/tuwiendsg/SmartCom/wiki
[3] http://cmmiinstitute.com/

re-negotiations). As we are interested in elasticity control operations management, we focus on *Elasticity Responsibilities* (Eq. 9.2) related to elasticity dimensions [36].

$$Roles = \{(R, Aut) | R \in Responsibilities, Aut \in [min, max]\} \tag{9.1}$$

$$Responsabilities = \{x \vee Relations(x, y) | x, y \in \{Cost, Quality, Resources, Error, Analytics\}\} \tag{9.2}$$

We define a *Message* as being composed of a `header`, and a `body`, the header containing `initiator` and `receiver` related information, while the body contains the message `type`, its `priority` and the `content` of the message. The message content (Eq. 9.4) can contain suggested interactions, which are nested interactions suggested by the initiator for the receiver(e.g., an elasticity controller can suggest the Procurement Analyst to re-negotiate the contract with the cloud provider due to cost increase). Using these entities, an *interaction* can be defined as a tuple of *Initiator-Receiver-Message* (Eq. 9.5), where each of the Initiator and Receiver can be a set of Roles.

$$\begin{aligned} Messages = \{[Header, Body] | \\ Header \in \{[MessageID, Initiator, Receiver] | \\ Initiator, Receiver \in \{Roles \cup ElasticityController\}\} \\ Body \in \{[MessageType, Priority, Content] \\ MessageType \in \{Notification, ChangeRequest, ErrorNotification\}\} \end{aligned} \tag{9.3}$$

$$\begin{aligned} Content = \{[Cause, SuggestedMeasure] | \\ Cause \in Requirements \cup ExpectedBehavior \\ SuggestedMeasure \in Interactions \cup Actions\} \end{aligned} \tag{9.4}$$

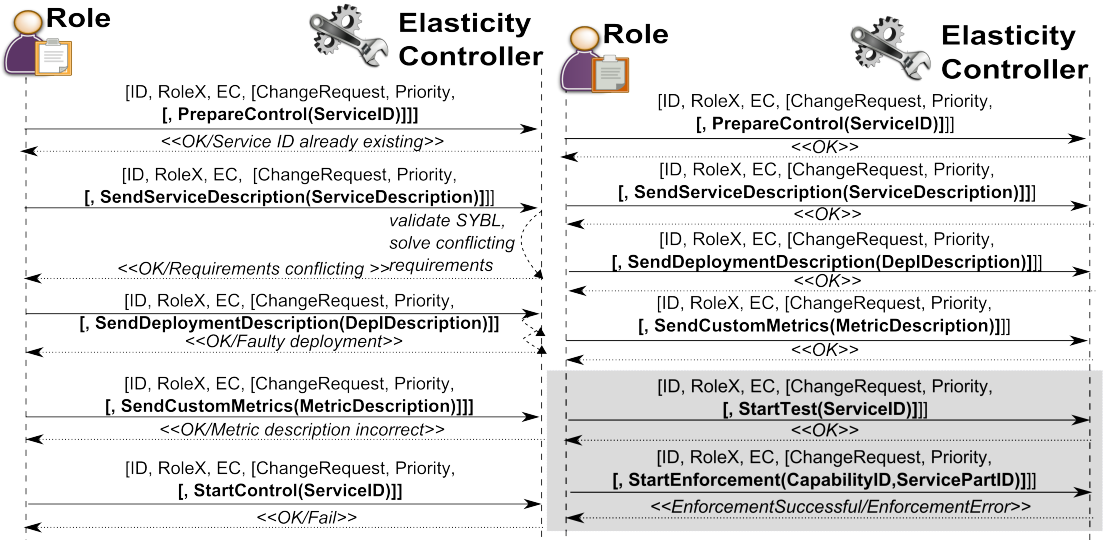
$$\begin{aligned} Interaction = \{[InteractionID, Initiator, Receiver, Message] | \\ Initiator, Receiver \in Roles, Message \in Messages\} \end{aligned} \tag{9.5}$$

The set of all interactions, which impact elasticity control, in an organization are defined as *OrganizationInteractions*, with the condition that at least one of the initiator and receiver are part of the organization.

$$\begin{aligned} OrganizationInteractions = \{Interaction | \\ Interaction.Initiator \vee Interaction.Receiver \in Organization\} \end{aligned} \tag{9.6}$$

9.4.2 Interaction protocols for supervisory control of elasticity

As discussed previously, elasticity depends on a large set of variables, both from the IoT, cloud and business world. Elasticity behavior of a service is subject to the business



(a) Interactions for starting elasticity control (b) Interactions for testing elasticity capability

Figure 9.4: Interaction dialogs

Interaction type	Interaction details
Undeploy the service	`[ID, Role, EC, [RFC, Priority, [Cause, UndeployService(ServiceID)]]]`
Replace metric composition rules	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceRules(ServiceID, CompositionRules)]]`
Replace deployment	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceDeployment(ServiceID, DeplDescription)]]`
Replace elasticity requirements	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceRequirements (ServiceID, Requirements)]]`
Pause/Resume control	`[ID, Role, EC, [RFC, Priority, [Cause, PauseControl(ServiceID)]]`

Table 9.2: Interactions for requesting modification in control

strategy of the service provider, and this can vary with the economic perspectives, and with the market evolution (e.g., the financial manager should decide the strategy in case of financial crisis, and not the elasticity controller). We propose using a *supervisory control mechanism* [38, 112], in which any decision of the human overrides any decision of the elasticity controller (i.e., the roles are the outer control loop). For this, we define a set of interaction protocols, based on the entities defined above, for facilitating the communication between roles and elasticity controllers.

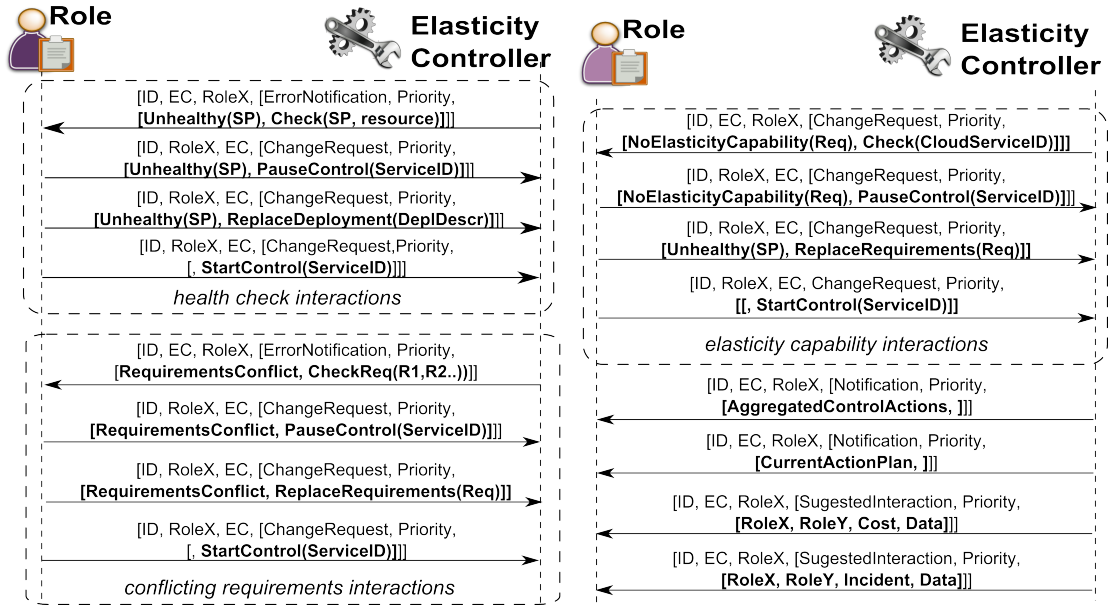


Figure 9.5: Elasticity controller bringing the roles into the control loop

Role as initiator - Bootstrapping Dialogs:

The goal of this interaction is to enable roles to initiate dialogs with the elasticity controllers for bootstrapping the elasticity controller (Figure 9.4a and 9.4b). For *starting elasticity control*, the elasticity controller is sent a `prepare` message, followed by information describing the cloud service is sent one at a time (e.g., service description, custom metrics description), and if each step is successfully achieved, a `Start Control` message is sent. Each call from the role to the dialog starts a complex process on the controller side (e.g., possible elasticity requirement conflicts are solved). For *testing an elasticity capability*, a `Start Test` message is sent for starting the test mode. For instance, the `System Administrator` is able through this dialog to set all needed information for elasticity control (e.g., structure, resources used), wait and ensure that each step is successfully completed.

Role as initiator - Request for change :

The goal of this interaction is to enable the roles to modify expected service behavior during runtime (Table 9.2). Whenever a role decides that an update is necessary, e.g., due to events signaled by the elasticity controller, the role can modify elasticity requirements, or deployment description (e.g., after a manual re-deployment). For instance, the `Service Manager` can decide to undeploy the service from the cloud environment, and, e.g., keep only an on-premise deployment.

Elasticity controller as initiator:

The goal of this interaction is to enable the elasticity controller to notify appropriate roles on changes in elasticity behavior (Figure 9.5). Whenever abnormal changes are observed in the cloud service behavior, the elasticity controller notifies roles, depending on the *Responsibilities*, and the *Authority* that they have associated. For instance, in the case of conflicting requirements that cannot be automatically solved (e.g., response time is expected to be low, while the cloud provider is running in degraded mode), the controller notifies the roles causing the conflict (i.e., $Role_i \ldots Rolej$), as well as a *higher authority role* having the responsibilities $\cup_{x=i..j} Responsabilities(Role_x)$.

9.4.3 Elasticity directives-driven interactions

For creating custom interactions, we have extended the SYBL [25] elasticity requirements definition language with the new `NOTIFY` directive, with BNF form described in the Listing 9.1, to be triggered when certain conditions hold. A call of the `notify()` method of the `NOTIFY` directive maps to the initiation of a new interaction between the elasticity controller and the role mentioned in the directive. An example of such a directive can be `No1: NOTIFY OperationsManager WHEN responseTime > 1.2 s : notify(WARNING, "Response time exceeds 1.2 s")`. Whenever a condition for a notification directive is true, the *Controller Communication* starts an interaction (i.e., translating the aforementioned directive into interaction `[No1, EC, Operations Manager, Notification, "Response time exceeds 1.2 s"]`.

Listing 9.1: SYBL Notification in Backus Naur Form (BNF)

```
Notification := notificationID:NOTIFY Role WHEN
   ComplexCondition
                        : notify(NotificationType, message)
Role := ROLE(Responsability1, Responsability2), Role |
        ROLE (Responsability1, Responsability2)|
        RoleX, Role | RoleX
NotificationType := NOTIFICATION | ERROR | WARNING
```

9.4.4 Interaction Aggregation

Each role, depending on its responsibilities, should receive different number of messages, or only emergency messages, the amount of messages being inversely proportional with the authority and directly proportional with the responsibilities (e.g., a Service Manager role with an authority of 10 out of 10 should receive less often messages than the System Administrator with authority 5 out of 10). Moreover, the nature of the messages should reflect the responsibilities and interests. For this, the *Controller Communication Module* examines messages and identifies metrics of interest for the responsibilities associated with each role. For filtering the interactions initiated, we define an aggregation function can be defined, selecting the amount of messages to be sent to the roles. 9.7 shows the

logarithmic *filtering function* we defined for interaction aggregation, which decides if the aggregation of messages so far should be sent.

$$f(Role, QInteractions) = (\log_{10}(Role.Authority) * THRESHOLD\_NOTIFICATION <= QInteractions.size()) \vee (\log_{10}(Role.Authority) * THRESHOLD\_ERROR <= MaxPriority(QInteractions)) \quad (9.7)$$

9.4.5 Message mapping

Messages are generated by the *Control Communication* component, based on metrics encountered in the message generated by the controller, and their mapping to the role responsibilities. We provide a generic processing mechanism that searches metric patterns associated with responsibilities in the message from the controller, and creates a new message with the structure described in Section 9.4.1, initiating interactions for the appropriate roles, considering roles' responsibilities. Depending on roles authorities (Equation 9.2), interactions are either aggregated or immediately sent to the *Interaction Management* component. This component publishes interactions to be observed in the eOMP platform, and in a *Collective Communication* component (e.g., SmartCOM) that enables complex interaction management among service provider team members. Although the team communication management is out of the scope of this thesis, service elasticity can be impacted by coordination mechanisms and information sharing, which is managed through the *Collective Communication* component.

9.5 Prototype and Experiments

9.5.1 Prototype

The elasticity Operations Management Platform (eOMP) is implemented as a Java enterprise application, which can be deployed either in the cloud or under service provider's premises. eOMP is open-source and available together with further experiments, details and user guides[4]. The current version integrates with the rSYBL elasticity controller, making use of the notification queue (i.e., embedded queue in the eOMP design, implemented using ActiveMQ[5]) exposing events during runtime. This can be easily extended to other elasticity controllers, by implementing an adapter for receiving and processing events. The service queue is using CloudAMQP[6], which is managed RabbitMQ[7] offered as a cloud service. Our Primefaces[8]-based frontend includes dynamically generating diagrams and charts for the cloud service provider employees.

[4] http://tuwiendsg.github.io/rSYBL/eOMP
[5] http://activemq.apache.org
[6] http://cloudamqp.com
[7] http://rabbitmq.com
[8] http://www.primefaces.org/

9.5.2 Elasticity Operations Management Features

To illustrate eOMP features, we used our pilot application[9] that consists of: (i) an event processing topology composed of an event processing unit and a load balancer, and (2) a data end topology composed of a data node unit and a data controller unit. We used a recording of a previous run, to which we injected events (i.e., by modifying monitored data for a limited amount of time). We chose this approach instead of real-time injecting faults, since it is more reliable, and our focus is showcasing the eOMP platform, and not the service versatility. Figure 9.6 shows the initial view of eOMP platform, with currently controlled services, as well as general information like current roles of service provider employees, their responsibilities, the dialogs that can be initiated using eOMP.

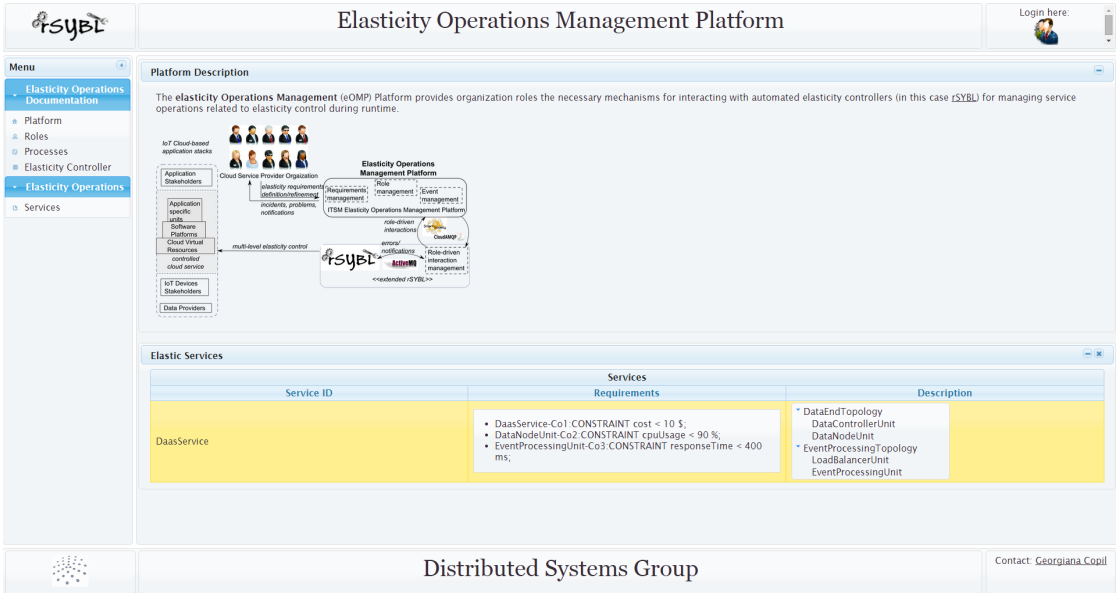


Figure 9.6: eOMP snapshot: initial information

Understanding responsibilities

After logging in, the employee can see his/her associated roles and responsibilities (Figure 9.7, as well as the roles and responsibilities of other team members. For adding new employees to eOMP, an administrator can select, according to the requirements, roles associated with the new employee.

Implicit vs. explicit interactions

For understanding current service behavior, roles need elasticity controller interactions executed regularly (e.g., every 10 minutes, or each time the role logs into the platform).

[9] https://github.com/tuwiendsg/DaaSM2M

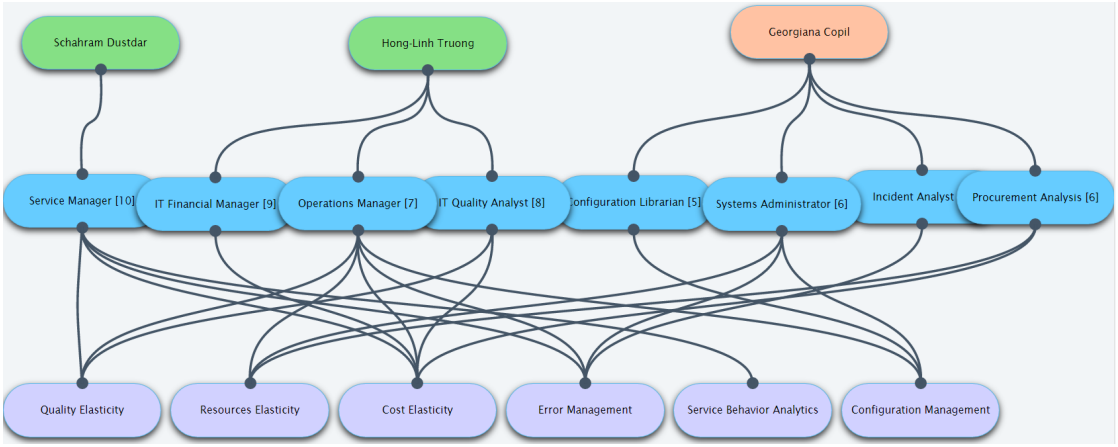


Figure 9.7: eOMP snapshot: current roles and responsibilities

We distinguish between two types of interactions from the user/employee perspective: (1) implicit interactions, for getting the necessary data to be displayed to the employee (e.g., dialogs for getting the service description), and (2) explicit interactions, initiated by the eOMP user. Figure 9.8 shows a dialog from the first type, with the system administrator (one role of the current logged in employee) requesting for initial description information. While without eOMP, the employee would need to manually call them, with eOMP the implicit interactions are already managed when the employee logs on in the platform.

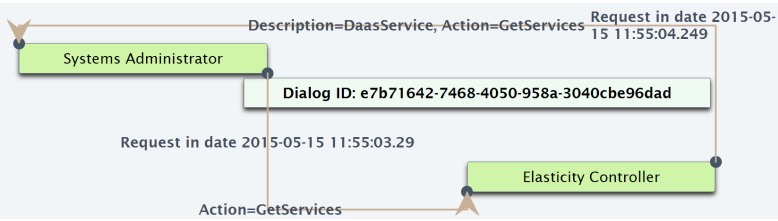
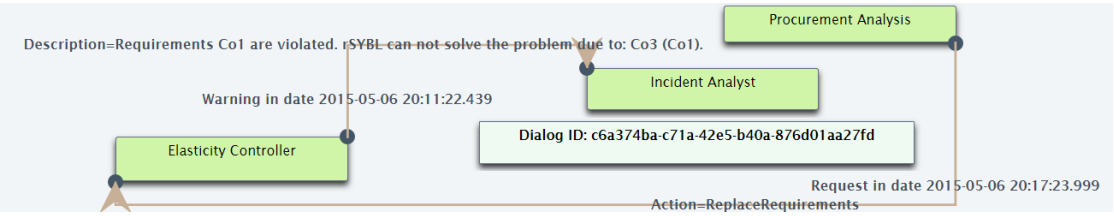


Figure 9.8: eOMP snapshot: implicit initial dialog requesting services information

Solving conflicting requirements

The controller can encounter a case where no actions are suitable for solving the discovered issues. The employee then can try to manually inspect the issue, to re-deploy, provide further elasticity capabilities, re-configure some artifacts, and then notify the appropriate roles(e.g., Configuration Librarian for the case configurations were performed), including the Elasticity Controller. Figure 9.10 shows a situation where constraints `Co1`("Co1:CONSTRAINT cost<10$;") and `Co3`("Co3:CONSTRAINT responseTime<400 ms;") are conflicting, because of the high workload and the limit on the cost. Since the employee receiving this interaction has more roles associated, s/he decides to replace the requirement from the `Procurement Analyst` role, for being

able to increase the limit for the service cost. The interaction dialog from the three roles is shown in Figure 9.9a, and consists of two steps: (1) the controller notifies the `Incident Analyst` role that no action is available due to the requirements conflict, (2) the employee uses the `Procurement Analyst` role to modify the cost requirement. Moreover, eOMP uses knowledge on role types and their authorities, avoiding modification conflicts (e.g., with no eOMP, two roles can fix observed issues at the same time). In our case, the role interaction with the elasticity controller is managed by eOMP, the employee being able to choose even a different role from which the issue can be solved better.



(a) eOMP snapshot: dialog for clarifying requirements

Figure 9.9: Conflicting requirements resolution

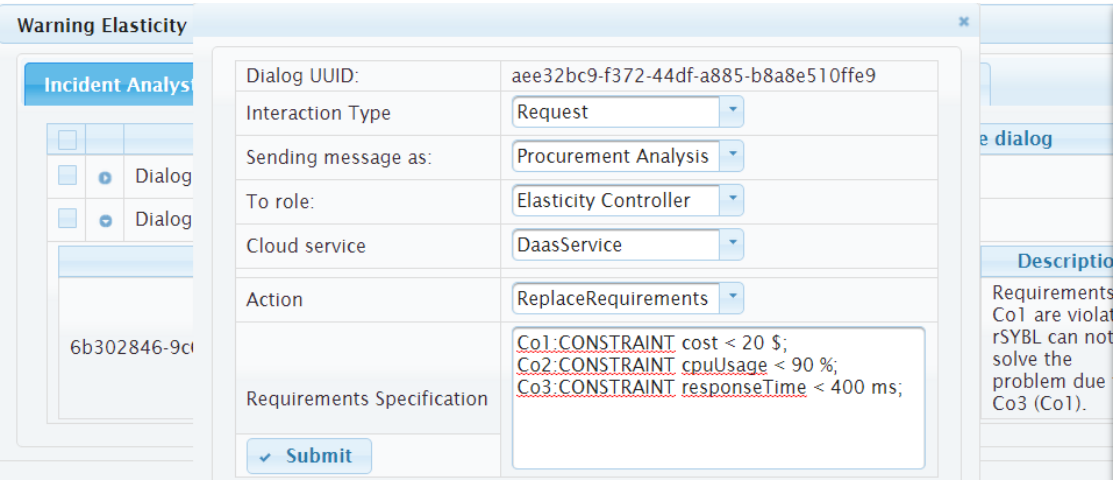


Figure 9.10: eOMP snapshot: replace requirements

Service health incidents

Another issue that may occur during service operation is that a service part might be unhealthy (i.e., monitoring metrics have error-like values for an amount of time). The controller exposes a notification with regard to this behavior, and the `Controller Communication` eOMP component processes it and initiates interactions with the appropriate roles (between the Elasticity Controller and the Operations Manager).

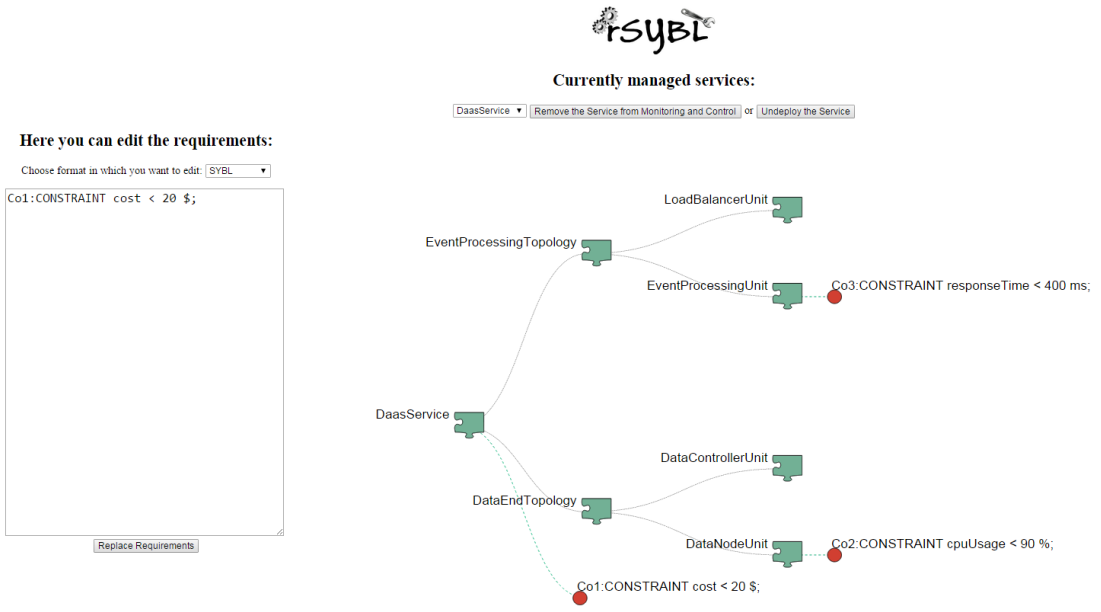


Figure 9.11: eOMP snapshot: requirements modified in rSYBL controller

Figure 9.12 shows a dialog that occurs when the `Event Processing Topology` is unhealthy (i.e., its metrics have error values).

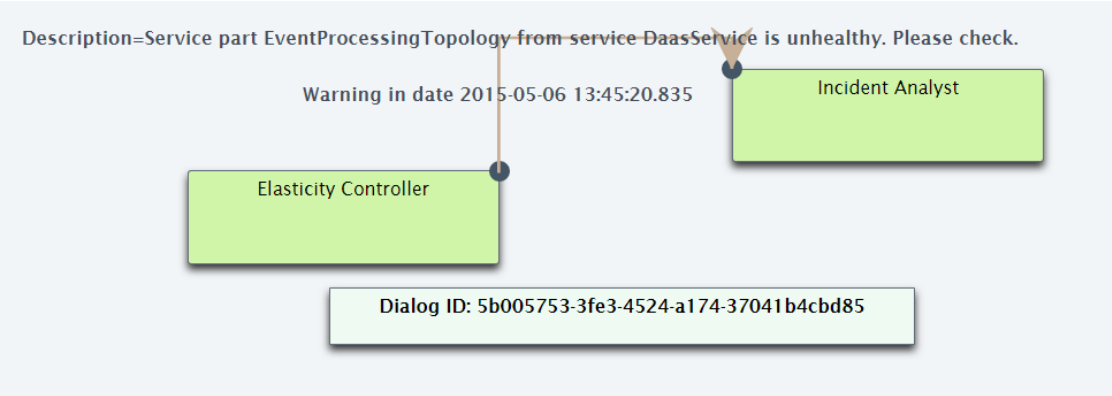


Figure 9.12: eOMP snapshot: unhealthy service part notification

With eOMP, all the roles that have incidents as responsibilities receive notifications, but the timing and the amount of notifications is inversely proportional with their authority. When the Operations Manager (high authority) gets an interaction in this sense(Figure 9.14 `Event Processing Topology` incident), it means that the other lower level authority roles have ignored or weren't able to address the situation. Therefore, this role can choose to delegate the interaction to the Incident Analyst (see Figure 9.13.

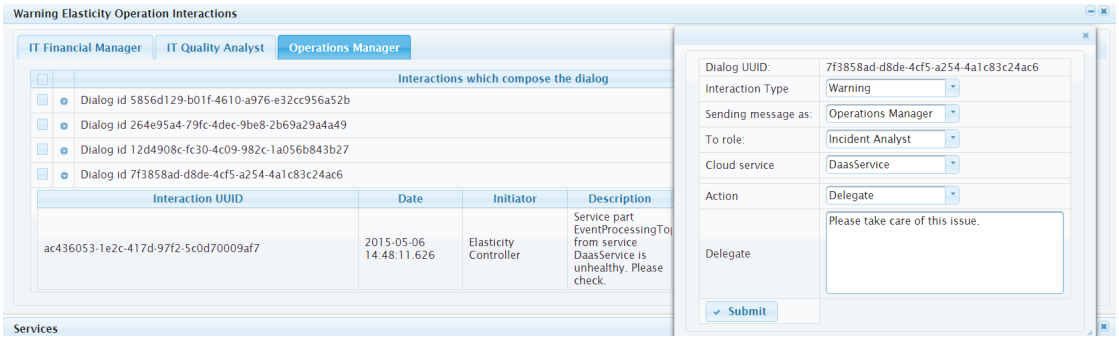


Figure 9.13: eOMP snapshot: unhealthy service part notification

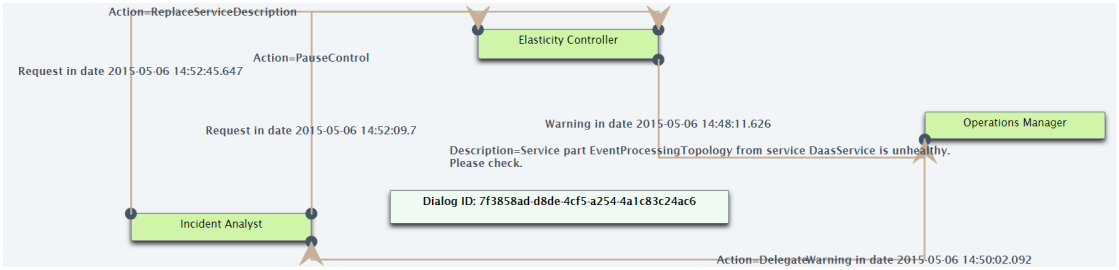
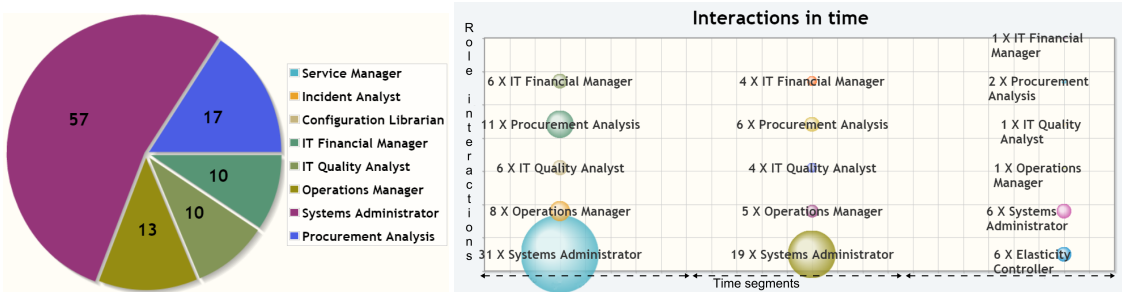


Figure 9.14: eOMP snapshot: unhealthy service part dialog

The incident analyst can try to fix the issue, or can report on the difficulty of the issue. Figure 9.14 shows the interactions (i.e., pause-fix-replace service description) undertaken by the Incident Analyst. All this is part of the initial dialog, so it can now be followed by the Operations Manager, to make sure the incident is being solved.



(a) Total role interactions (b) Role interactions in time

Figure 9.15: eOMP snapshots: statistical information regarding interactions

Dealing with roles authorities

eOMP enables highest authority roles to follow interactions that occurred over time and how they are distributed among roles (see Figure 9.15). The amount of messages that a

role receives varies with the events that occur, with role's responsibilities in relation with these events, and with role's authority. We can see that higher level authority roles have fewer interactions, following the Interaction aggregation function in Section 9.4.4.

Thus, eOMP facilitates interactions between service providers and elasticity controllers, and among service provider roles, automating operation tasks and providing support for interaction management based on role's authority and responsibilities. With eOMP the roles can easily follow the evolution of their elastic service, and the evolution, in time, of incidents, requests for change, or measures taken by the elasticity controller in order to control their service behavior.

CHAPTER 10

Related Work

10.1 Elasticity Requirements Language

Resource re-allocation and requirements specification have been targeted usually from the SLA fulfillment or scheduling and resource allocation perspectives. Macias et al. [79] provide an evaluation of possible SLA administration strategies, showing how cloud providers revenues change when optioning for different strategies. Fard et al. [39] approach static scheduling with a different view, defining a multi-objective optimization algorithm and demonstrating its usefulness on real-world applications. Han et al. describe in [56] an approach for fine-grained scaling at resource level in addition to the VM-level scaling, which uses a lightweight scaling algorithm for improving resource utilization while reducing cloud providers' costs. Our approach differs from these works in two main points: we support (i) multiple levels of elasticity control using (ii) multiple elastic properties.

In [85] the authors present an attempt to tackle the problem of elasticity from the point of view of resource and elasticity in SaaS based clouds. The authors propose relating cost with quality: cost per performance metric (C/P) and cost per throughput (C/T). Li et al. [77] review the existent metrics for evaluating commercial cloud services from economics evaluation metrics to elasticity evaluation metrics setting the focus on the complexity of cloud computing environments. Sharma et al. [109] propose a framework for cost optimization, considering the fact that the resources, the cost and the quality obtained influence each other. Therefore, the idea of application elasticity in cloud as a complex multi-dimensional problem is not new, and research effort goes into providing solutions or partial solutions. However, existing works have not developed flexible languages for controlling multi-dimensional elastic properties.

In [47] Galan et al. propose an extension of OVF for service specification in cloud environments describing resource as well as business rules and enforcing them through resource allocation/de-allocation. Chapman et al. [19] describe an elastic service definition in computational grids. Morán et al. [91] provide a rule-based approach for specifying

application requirements. In contrast with these two approaches, our main focus is describing elasticity requirements and the different granularities at which they can be specified by developers, end-users or cloud providers.

From the initial publication of SYBL [25], several languages have been proposed for aiding elasticity in the cloud. Kouki et al. [69] propose an extension of CSLA (Cloud Service Level Agreement) [20], focusing on QoS degradation and penalty models for an easier and clearer interaction between the cloud customer and the cloud provider. Li et al. [76] propose PSLA (PaaS level SLA) language for the description of SLA, focusing on workload elasticity and PaaS-specific properties. PSLA is an extension of WS-Agreement [5]. CloudMF [42] is a language for cloud infrastructure resources management, with focus on uniformly describing applications for reducing vendor lock-ins. Based on CloudML [53], Kritikos et al. [71] propose SRL, a policy language for scaling multi-cloud applications. Zabolotnyi et al. [131] propose SPEEDL, a declarative language for event-based scaling of cloud applications.

The major difference between existing work and our approach is that our work tackles elasticity requirements from more than one perspective (resource, quality, cost) and at different levels of granularity, thus assigning the user the capacity of specifying when the application should scale throughout its execution and, most importantly how.

10.2 Elasticity Control Mechanisms

Yang et al. [129] support just-in-time scalability of resources based on profiles. Kazhamiakin et al. [66] consider KPI dependencies when adapting the service based applications. Malkowski et al. [80] support multi-level modeling and elastic control of resources for workflows in the cloud. Guinea et al. [55] develop a system for multi-level monitoring and adaptation of service-based systems by employing layer-specific techniques for adapting the system in a cross-layer manner. Elasticity control of storage based on resources and quality has been focused by various research work, e.g., adaptively deciding how many database nodes are needed depending on the monitored data in [121]. In [127] the authors propose an algorithm for software resource allocation considering the loads, analyzing the influence of software resources management on the applications performance. Yu et al. [130] propose an approach for resource management of elastic cloud workflows. They present a generic workflow architecture with components such as makeflow (that paralelizes large complex workflows on clusters grids and clouds) and master-work-workers. Almeida et al. [4] propose a branch and bound approach for optimally selecting services from multiple clouds during runtime. In [70] the authors propose a framework for automatic scalability using a deployment graph as a base model for the application structure and introduce elasticity as a service cross-cutting different cloud stack layers (SaaS, PaaS, IaaS).

From the initial publication of rSYBL mechanisms, several elasticity controllers have been proposed. Tolosana-Calasanz et al. [118] propose controlling resources necessary for data streams, using a shared token bucket approach. Dupont et al. [33] propose the notion of software scalability, both horizontal and vertical, by drawing inspiration

from infrastructure scalability. For SaaS providers, horizontal (i.e., adding/removing more software units), or vertical (i.e., increasing/decreasing offerings for the service), can be a good customization opportunity to profit from elasticity at IaaS level. Aragna et al. [7] define metrics and rules for elasticity control, and study various scenarios (e.g., infrastructure only, or database only control). Nakos et al. [93] propose an approach for elasticity control based on dynamic instantiated Markov decision processes, using probabilistic model checking. Ferry et al. [41] propose an approach for the continuous management of scalability in multi-cloud systems, based on ScaleDL [12] and CloudMF [42].

Compared to the above-mentioned work, we control elasticity not just in terms of resources but also in terms of quality and cost and use application structure for proposing an accurate multiple level control of elasticity of cloud services. Furthermore, they lack user-customized elasticity control. We propose a user oriented elasticity control in which the user (cloud service creator, application developer, etc.) specifies how the cloud service should behave for achieving the elasticity property. Moreover, we argue in favor of an elastic services control aware of the structure of the elastic service, profiting from this knowledge for a multiple levels elasticity control of cloud services.

10.3 Elasticity Behavior Estimation

In our previous work, we focused on modeling current and previous behavior with the concepts of *elasticity space* and *pathway* [89], where we utilize different algorithms to determine enforcement times in observed service behavior (e.g., with change-point detection), but without modeling expected behavior of different service parts, in time. Verma et al. [124] study the impact of reconfiguration actions on system performance. They observe infrastructure level reconfiguration actions, with actions on live migration, and observe that the VM live migration is affected by the CPU usage of the source virtual machine, both in terms of the migration duration and application performance. The authors conclude with a list of recommendations on dynamic resource allocation. Kaviani et al. [65] propose profiling as a service, to be offered to other cloud customers, trying to find tradeoffs between profiling accuracy, performance overhead, and costs incurred. Zhang et al. [134] propose algorithms for performance tracking of dynamic cloud applications, predicting metrics values like throughput or response time.

Dean et al. [30] propose an approach for predicting running application performance anomalies, self-organizing maps for capturing emergent behavior and predicting unknown anomalies. Using unsupervised learning, this approach also identifies previously unknown anomalies/faults that may appear in the system (e.g., memory leaks, cpu leaks). For cloud service SLA violation prediction several solutions have been proposed, such as [74] [116], that use statistical models (e.g., decision trees, artificial neural networks) or naive bayes classifiers, predicting when the SLA would be violated without focusing on the violation cause. LaCurts et al. [72] propose Cicada, a framework that predicts network traffic for cloud applications, without making assumptions about application structure. The authors argue that cloud providers should offer predictive guarantees as

a service, instead of bandwidth guarantees, which would also encapsulate application runtime changes. Similarly, ADVISE-enabled rSYBL can be used to guarantee or to sell as a service cloud service elasticity, with little specifications coming from the user.

Juve et al. [64] propose a system that helps automating the provisioning process for cloud-based applications. They consider two application models, one workflow application and one data storage case, and show how for these cases the applications can be deployed and configured automatically. Li et al. [75] propose CloudProphet framework, which uses resource events and dependencies among them for predicting web application performance on the cloud. Shen et al. [110] propose the CloudScale framework that uses resource prediction for automating resource allocation according to service level objectives (SLOs) with minimum cost. Based on resource allocation prediction, CloudScale uses predictive migration for solving scaling conflicts (i.e. there are not enough resources for accommodating scale-up requirements) and CPU voltage and frequency for saving energy with minimum SLOs impact. Compared with this research work, we construct our model considering multiple levels of metrics, depending on the application structure for which the behavior is learned. Moreover, the stress factors considered are also adapted to the application structure and the elasticity capabilities (i.e. action types) enabled for that application type. Cuomo et al. [29] propose a methodology for performance and cost estimation for mOSAIC [105] cloud applications, under generic workloads.

Compared with presented research work, we focus not only on estimating the effect of an elasticity control process on the service part with which it is associated, but on different other parts of the cloud service. Moreover, we estimate and evaluate the elasticity behavior of different cloud service parts, in time, because we are not only interested in the effect after a predetermined period, but also with the pattern of the effect that the respective ECP introduces.

10.4 Heterogeneous Elasticity Control

In multi-cloud area ongoing research puts great emphasis on federation, from two perspectives: multi-cloud service controllers that assume that they are using federated clouds, e.g., Bermbach et al. [13], Kondikoppa et al. [68], Panda et al. [101], and work done for the federation of clouds e.g., Kertesz et al. [67], Paraiso et al. [102]. In our work, we are not dealing with a federation of clouds, which implies some sort of standardization, and neither with already federated clouds. From a practical point of view, the controllers of such multi-cloud services need to acknowledge the differences among these clouds, especially in cases in which the clouds need to be heterogeneous by the nature of service parts requirements and functionalities.

Sampaio et al. [108] propose Uni4Cloud, a framework for the deployment and management of multi-cloud services. They emphasize the challenges in deploying and configure such a service, and present a model-based approach for accomplishing these tasks, automatically and independent of the IaaS provider. However, the focus in this case is the deployment, not runtime elasticity. Miglierina et al. [88] propose a control theoretic approach for multi-cloud services, targeting resource level control and modeling formally

specific types of service units that are supported (e.g., load balancer), and putting a lot of focus on the load balancing between instances of the same components deployed on different cloud infrastructures. As opposed to this, we aim supporting all types of services, showing our customization mechanisms for providing elasticity control, and we focus on cloud services that deploy different service topologies or service units on heterogeneous clouds, due to their difference by nature (e.g., gateways versus web services, backup service unit instances versus normal ones). Aragna et al. [8, 9] propose a model-driven approach for the multi-cloud deployment and monitoring of cloud services over multiple cloud infrastructures, describing a framework to support full transparency from the user. We argue that, in fact, user transparency is much more relevant during runtime, where the user should be able to easily describe high-level, provider-independent requirements, with the support framework controlling, transparently, the service.

In contrast with other approaches, our work focuses on the heterogeneity of clouds and their services, thus giving better control to service developers on the elasticity achieved during runtime. Moreover, we propose maintaining the single cloud requirements structures, thus shifting only the users' focus towards the elasticity metrics they are interested in, specific for multi-cloud services. This way, users can profit from differences among cloud providers, and improve their service performance or achieve complex services with the combination of highly domain-specific private clouds and public clouds.

10.5 Elasticity Operations Management

Various standards and processes have been proposed over the years for IT service management. Sallé [107] provides an analysis of the evolution of IT service management over the years, and its evolution towards IT governance. Starting from Information Systems Management Architecture [122], IT management methodologies have evolved towards well-defined standards/best practices of IT service management (e.g., ITIL® [10], BSI ISO 20000 [17], FitSM [40]). Although the focus of this thesis is not the management processes adopted by organizations, understanding their internal processes is necessary for being able to support them in their quest for cloud service elasticity. Since the latest reference models used nowadays in organizations (e.g., [17], [58]) are in alignment with ITIL® service management practices, we used these processes and organization roles. However, our design (see Section 9.4) is such that, roles and processes can be modified for accommodating future service management models.

Operation management in cloud computing has been approached mostly from the cloud provider's perspective [59,133], and little from service provider (i.e., cloud customer) perspective. Bleizeffer et al. [15] propose a set of user roles in cloud systems, having as core roles not only the cloud service provider, but also the cloud service consumer and cloud service creator. The three core roles are expanded into a taxonomy of interconnected user roles, which communicate with each other for delegating responsibilities or gathering information. Demont et al. [31] present an initial proposal of integrating the TOSCA cloud service description standard with ITIL elements. Several commercial solutions enable cloud infrastructure management support, but do not support service operations

management at cloud customer's service level (e.g., VMWare vCloud Suite[1], Oracle Enterprise Manager[2]). Liu et al. [78] propose an incident diagnosis approach based on incident relationships, using co-occurring and re-occurring incidents for performing root cause analysis. Munteanu et al. [92] propose an architectural approach for incident management in the cloud, from service monitoring perspective, including incident lifecycle management, event and incident detection, incident classification and recovery and root cause analysis.

In contrast with above presented work, we focus on the elasticity aspect of service operations management in the cloud, characterizing the relevant properties and interactions. Moreover, we emphasize the importance of supervisory control for the cloud, and introduce service provider employees as first-class entities in the control loops.

[1] http://www.vmware.com/products/vcloud-suite

[2] http://www.oracle.com/technetwork/oem/enterprise-manager

CHAPTER 11

Conclusions and Future Work

This chapter summarizes the main results of the thesis, emphasizing the thesis contributions. Furthermore, future research directions are outlined, based on research presented in this thesis and considering challenges that were not completely addressed due to the limited time frame of the thesis.

11.1 Conclusions

The thesis is based on the observation that cloud service stakeholders need expressive ways of describing their requirements, and they need tools to support them to achieve these requirements. The *SYBL language* enables a wide range of users, from developers to cloud customers and cloud providers, to specify the service's elasticity in a simple way, while the actual complex enforcement of elasticity remains transparent to the users. SYBL features, covering different elasticity constraints, strategies and monitoring directives, permit a wide range of flexible ways for controlling service's elasticity, while for cases where some needed metrics are not yet present in SYBL, the language can be easily extended to reflect the elastic properties of user's focus.

After developing the language, the framework enforcing these requirements was developed. *rSYBL*, the framework for multi-level cloud service elasticity control, considers multi-level elasticity requirements coming from the cloud service provider. rSYBL framework is open-source, extensible, and can be customized for different cloud providers and cloud services, having various preferences in terms of elasticity control. We base our control mechanisms on the user-provided requirements, and on cloud service pre-deployment information and runtime information. This way, we empower the users to steer the control by specifying their needs, at the service level they possess knowledge for (e.g., in the case s/he is aware of how or what should be controlled at the data end but not at the business end), and the level of detail that they are comfortable with (e.g., if the cloud provider knows only requirements about cost in relation to the number of clients they expect, s/he is not needed to specify response time requirements).

We observed how rSYBL behaves under various use-cases, and proposed a complex use-case with cloud services deployed in multiple, heterogeneous clouds, with dependencies among various service parts. The need for elastic multi-cloud services was emphasized, showing the main challenges, and presenting models and mechanisms for providing heterogeneous multi-cloud control for services. rSYBL framework was extended, with these mechanisms, and some experiments were presented with a service deployed on two different cloud infrastructures, with different time sensitivities.

For better elasticity control, the rSYBL controller, or any other elasticity controllers, need to understand how their service behaves as a whole, how different parts behave, in time, in relation to the elasticity requirements. A methodology for *estimating cloud service elasticity behavior* was presented, and implemented in the ADVISE framework. ADVISE is able to estimate the behavior of cloud service parts, in time, when enforcing various $ECPs$, by considering different types of information represented through the elasticity dependency graph. Experiments from three different cloud services, show that ADVISE framework is indeed able to *advise* elasticity controllers about cloud service behavior, contributing towards improving cloud service elasticity. ADVISE was integrated in the rSYBL elasticity controller [24], and the decision mechanisms we refined in order to consider continuous ECP effects. The improvement that ADVISE brings to the rSYBL elasticity controller was highlighted, and various decision types that ADVISE influences were discussed.

After developing the mechanisms above, for controlling the elasticity of cloud services, we considered the human factor that influences the control process. The elasticity controller receives requirements from stakeholders (e.g., cloud service owner, cloud service developer), in order to control the elasticity of the cloud service at runtime. However, normally, cloud services are developed and operated in an organization, by various employees with various responsibilities. This thesis presented a set of interaction protocols for managing elasticity operations of cloud services, which take into consideration service provider roles as first class entities in the service elasticity control. The eOMP platform was introduced, which allows service provider employees to manage the cloud service operations related with elasticity, interacting with the elasticity controller and other employees of the service provider.

Some of the presented prototypes consider the base of other prototypes, or research work. SYBL is used in the Eclipse project Cloud Application Management Framework (CAMF[1]) as a specification language for describing elasticity requirements. rSYBL is a core component of CELAR project[2], functioning as the CELAR decision-maker, controlling cloud applications that are using the CELAR platform. rSYBL and SYBL were extended to be used in IoT governance, for specifying and managing governance policies for IoT applications [95]. Moreover, rSYBL was used as a basis for research on elasticity coordination mechanisms [83], and on porting applications to the cloud [117].

[1] `https://projects.eclipse.org/proposals/cloud-application-management-framework`

[2] `http://www.celarcloud.eu`

11.1.1 Research Questions Revisited

We revisit the research questions in Section 1.2, and show how and to which extent we have addressed them:

- ***Research Question 1: How could requirements be specified, with the least effort from service provider side, but still giving sufficient information for controlling the service?***
  We address this question by proposing the SYBL language (Chapter 5), which allows users to specify high-level requirements at multiple levels of abstraction. The basis of the SYBL language is the model presented in Section 8.2, which links design-time and run-time information, in terms of requirements, used cloud resources, and system level, and application level metrics. SYBL requirements can be specified either in text format (e.g., in the rSYBL user interface, as TOSCA policies, or in the service description) or with the CAMF[1] user interface. Moreover, we extended the SYBL language for it to be able to represent requirements regarding uncertainty (e.g., uncertainty level the user is comfortable with, when applying a specific strategy) [95]. Due to its generic constructs, SYBL can be extended and used to other areas, as we discuss in next Section.

- ***Research Question 2: How can we control the elasticity of complex services deployed in the cloud?***
  To answer this question, we defined mechanisms for controlling cloud services, based on the model described in Section 8.2 and the language presented in Chapter 5. We formulated the problem of finding the set of actions necessary to fulfill requirements as a set coverage problem, and based on it we implemented the rSYBL framework (Chapter 6). We addressed issues such as conflict resolution for requirements, and currently support a number of monitoring tools (e.g., MELA [90], Ganglia[3]) and enforcement mechanisms for different clouds (e.g., OpenStack[4], Flexiant FCO[5], Google Compute Cloud[6]). For understanding the impact, *in time*, of various control actions upon the service, at different levels of abstraction, in Chapter 8 we proposed a methodology based on clustering multi-dimensional points. We showed that using this mechanisms instead of initial manual profiling of the cloud service, rSYBL is much more stable (i.e., reduced "ping-pong"/control oscillation effect).

[3] http://ganglia.sourceforge.net
[4] https://www.openstack.org
[5] https://www.flexiant.com/flexiant-cloud-orchestrator/
[6] https://cloud.google.com/compute

- ***Research Question 3: How could elasticity control be integrated in service operations management?***
  We have shown in Chapter 9 that operations management is important for services running in the cloud, and that due to possibly complex interactions with elasticity controllers, support is needed for better integration of cloud elasticity aspects into operations management. We answered this question by proposing a methodology for integrating the elasticity control into operations management, considering that services are developed and managed in companies with many employees, each with various responsibilities. With this as a basis, we can further explore how elasticity-related events (e.g., incidents, or request for change) affect the service behavior, and study their correlations for understanding how to better control the service, and how to improve it from design-time to operation-time.

11.2 Future Work

For future work, we envision exploiting the rSYBL framework in research projects and collaborations, extending the framework, or mechanisms used, for addressing remaining issues or for adapting it to other domains. The following research directions are feasible for future work:

- We plan to extend rSYBL to support frameworks in IoT domain, such as Leonore [125]. rSYBL is highly customizable, and control actuators or enforcement mechanisms can be easily plugged in using already developed mechanisms. For managing IoT applications, we need to extend the base model that now is able to represent mainly IaaS cloud offered services, to be able to represent devices and sensors, and complex environments built from them (e.g., smart cities). Furthermore, the monitoring and analysis mechanisms need to be adapted to IoT domain, requiring more fine-grained error monitoring, and, of course, adapting decision types to more static, reconfiguration based ones, since new sensors/devices can be physically added only through human intervention.

- Study the extent to which the controller can be used for complex services, and, extend it to support more extreme cases (e.g., VMs hosting multiple Docker containers, each hosting a service unit or topology, and having various placement or privacy requirements). The model and controller were design before the burst in popularity of container-based services. Even though the model and the controller are designed to support multiple units per VM, a case study is necessary, with an even more complex services, with multiple units, multiple Docker containers per VM, and multiple elasticity capabilities, where we evaluate the challenges present, and extend rSYBL to deal with such cases. For this, we are looking into what makes cloud services elastic, and how many elasticity capabilities a cloud service should have (i.e., where is elasticity necessary and where it is overhead).

- Uncertainty[7] is very present in the cloud computing domain, due to its very dynamic nature. There is a high degree of uncertainty about service performance measurements, cloud providers availability, resource provisioning/de-provisioning timeliness and quality of the provisioned resources. To these, it adds environment-related uncertainties, e.g., whether the data center is available or not during flooding/storms or human actions. Such uncertainties should be considered by elasticity controllers, focusing on the ones that can be tackled from the cloud customer perspective. rSYBL will be used in the U-Test EU project[8], for understanding and tackling uncertainties.

- For managing large scale, complex services, organizations usually use teams, with employees having various roles, as we have seen in Chapter 9. eOMP can be extended in order to better support more types of events (e.g., various types of incidents, problems, differentiating between incidents and problems). Moreover, eOMP should be extended with mechanisms understanding similarities between decisions and events, for advising both the controller and the organization to better control the elasticity of the cloud service. eOMP can also be extended in order to support even multiple organizations simultaneously, operating the same or different services. For this, inter-organization interactions should be supported, and various issues may arise like understanding the overall role that the various organization play for the service operation.

[7] A taxonomy of uncertainties is presented in the tech report by Nastic et al. [114]
[8] http://www.u-test.eu

Bibliography

[1] 2014 State of the Cloud Survey. http://www.rightscale.com/blog/cloud-computing-trends-2014-state-cloud-survey.

[2] Ahmad Al-Shishtawy and Vladimir Vlassov. Elastman: Autonomic elasticity manager for cloud-based key-value stores. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 115–116, New York, NY, USA, 2013. ACM.

[3] Ahmad Al-Shishtawy and Vladimir Vlassov. ElastMan: Elasticity Manager for Elastic Key-value Stores in the Cloud. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 7:1–7:10, New York, NY, USA, 2013. ACM.

[4] A Almeida, F. Dantas, E. Cavalcante, and T. Batista. A branch-and-bound algorithm for autonomic adaptation of multi-cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 315–323, May 2014.

[5] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.

[6] Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. How to adapt applications for the cloud environment. *Computing*, 95:493–535, 2013.

[7] Claudio Ardagna, Ernesto Damiani, Fulvio Frati, Guido Montalbano, Davide Rebeccani, Marco Ughetti, et al. A competitive scalability approach for cloud architectures. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 610–617. IEEE, 2014.

[8] Danilo Ardagna. Cloud and multi-cloud computing: Current challenges and future applications. In *Proceedings of the Seventh International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*, PESOS '15, pages 1–2, Piscataway, NJ, USA, 2015. IEEE Press.

[9] Danilo Ardagna, Elisabetta Di Nitto, Parastoo Mohagheghi, Sébastien Mosser, C Ballagny, F D'Andria, G Casale, P Matthews, C-S Nechifor, D Petcu, et al. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *2012 ICSE Workshop on Modeling in Software Engineering (MISE)*, pages 50–56. IEEE, 2012.

[10] Valerie Arraj. Itil®: the basics. *Buckinghampshire, UK*, 2010.

[11] Prith Banerjee, Richard Friedrich, Cullen Bash, Patrick Goldsack, Bernardo Huberman, John Manley, Chandrakant Patel, Parthasarathy Ranganathan, and Alistair Veitch. Everything as a service: Powering the new information economy. *Computer*, 44(3):36–43, 2011.

[12] Matthias Becker, Steffen Becker, and Joachim Meyer. Simulizar: Design-time modeling and performance analysis of self-adaptive systems. *Software Engineering*, 213:71–84, 2013.

[13] D. Bermbach, T. Kurze, and S. Tai. Cloud Federation: Effects of Federated Compute Resources on Quality of Service and Cost. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 31–37, March 2013.

[14] Alexander Bird. Thomas Kuhn. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2011 edition, 2011.

[15] Terry Bleizeffer, Jeffrey Calcaterra, Deepa Nair, Randy Rendahl, Birgit Schmidt-Wesche, and Peter Sohn. Description and application of core cloud user roles. In *Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology*, CHIMIT '11, pages 2:1–2:9, New York, NY, USA, 2011. ACM.

[16] William F Brown and David H Hawkins. Remote access computing: the executive's responsibility. *Journal of Systems Management*, 1972.

[17] BSI Group. ISO/IEC 20000-Information technology-Service management. http://www.iso.org/iso/publication_item.htm?pid=PUB200013.

[18] Mark Carlson, Martin Chapman, Alex Heneveld, Scott Hinkelman, Duncan Johnston-Watt, Anish Karmarkar, Tobias Kunze, Ashok Malhotra, Jeff Mischkinsky, Adrian Otto, et al. Cloud application management for platforms. *OASIS, http://cloudspecs. org/camp/CAMP-v1. 0. pdf, Tech. Rep*, 2012.

[19] Clovis Chapman, Wolfgang Emmerich, Fermin G. Marquez, Stuart Clayman, and Alex Galis. Elastic service definition in computational clouds. pages 327–334, April 2010.

[20] EU Commission. Cloud Service Level Agreement. http://ec.europa.eu/digital-agenda/en/news/cloud-service-level-agreement-standardisation-guidelines.

[21] "ITSM Community". Itil roles descriptions. http://www.itsmcommunity.org/downloads/ITIL_Role_Descriptions.pdf.

[22] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[23] Georgiana Copil, Daniel Moldovan, Ioan Salomie, Tudor Cioara, Ionut Anghel, and Diana Borza. Cloud sla negotiation for energy savingâĂŤa particle swarm optimization approach. In *Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on*, pages 289–296. IEEE, 2012.

[24] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Multi-level elasticity control of cloud services. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *Service-Oriented Computing*, volume 8274 of *Lecture Notes in Computer Science*, pages 429–436. Springer Berlin Heidelberg, 2013.

[25] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Sybl: An extensible language for controlling elasticity in cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 112–119, May 2013.

[26] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. On controlling cloud services elasticity in heterogeneous clouds. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 573–578. IEEE, 2014.

[27] Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, and Marios Dikaiakos. Advise âĂŞ a framework for evaluating cloud service elasticity behavior. In Xavier Franch, AdityaK. Ghose, GraceA. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 275–290. Springer Berlin Heidelberg, 2014.

[28] Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. Supporting cloud service operation management for elasticity. In *Service-Oriented Computing*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015.

[29] Antonio Cuomo, Massimiliano Rak, and Umberto Villano. Performance prediction of cloud applications through benchmarking and simulation. *International Journal of Computational Science and Engineering*, 11(1):46–55, 2015.

[30] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In

Proceedings of the 9th International Conference on Autonomic Computing, ICAC '12, pages 191–200, New York, NY, USA, 2012. ACM.

[31] Christoph Demont, Uwe Breitenbücher, Oliver Kopp, Frank Leymann, and Johannes Wettinger. Towards integrating tosca and itil. In *ZEUS*, pages 28–31. Citeseer, 2013.

[32] DMTF. Cloud Infrastructure Management Interface (CIMI), October 2012.

[33] Simon Dupont, Jonathan Lejeune, Frederico Alvares, and Thomas Ledoux. Experimental analysis on autonomic strategies for cloud elasticity. In *2015 IEEE International Conference on Cloud and Autonomic Computing (ICCAC)*, 2015.

[34] S. Dustdar, Yike Guo, Rui Han, B. Satzger, and Hong-Linh Truong. Programming directives for elastic computing. *IEEE Internet Computing*, 16(6):72–77, 2012.

[35] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong Linh Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, 2011.

[36] Dustdar, Schahram and Guo, Yike and Satzger, Benjamin and Truong, Hong-Linh. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, 2011.

[37] Mica R Endsley. The application of human factors to the development of expert systems for advanced cockpits. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, number 12, pages 1388–1392. SAGE Publications, 1987.

[38] Mica R Endsley and David B Kaber. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42:462–492, 1999.

[39] Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo, and Thomas Fahringer. A multi-objective approach for workflow scheduling in heterogeneous environments. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 300–309, Washington, DC, USA, 2012. IEEE Computer Society.

[40] FedSM. FitSM: Standards for IT Service Management. http://www.fedsm.eu.

[41] Nicolas Ferry, Gunnar Brataas, Alessandro Rossini, Franck Chauvel, and Arnor Solberg. Towards bridging the gap between scalability and elasticity. In *CLOSER 2014: 4th International Conference on Cloud Computing and Services Science-Special Session on Multi-Clouds*, pages 746–751, 2014.

[42] Nicolas Ferry, Hui Song, Alessandro Rossini, Franck Chauvel, and Arnor Solberg. Cloudmf: Applying mde to tame the complexity of managing multi-cloud applications. In *UCC 2014: 7th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 269–277. IEEE Computer Society, 2014.

[43] Christiane Floyd. Software development as reality construction. In Christiane Floyd, Heinz Züllighoven, Reinhard Budde, and Reinhard Keil-Slawik, editors, *Software Development and Reality Construction*, pages 86–100. Springer Berlin Heidelberg, 1992.

[44] Heinz Foerster. On constructing a reality. In *Understanding Understanding*, pages 211–227. Springer New York, 2003.

[45] Forbes. Roundup of cloud computing forecasts and market estimates. "http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/", 2015.

[46] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13, 2009.

[47] Fermín Galán, Americo Sampaio, Luis Rodero-Merino, Irit Loy, Victor Gil, and Luis M. Vaquero. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, COMSWARE '09, pages 19:1–19:12, New York, NY, USA, 2009. ACM.

[48] Stuart D. Galup, Ronald Dattero, Jim J. Quan, and Sue Conger. An overview of it service management. *Commun. ACM*, 52(5):124–127, May 2009.

[49] A Gambi, D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar. On estimating actuation delays in elastic computing systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on*, pages 33–42, May 2013.

[50] Gartner. "forecast: Public cloud services, worldwide, 2013-2019, 2q15 update". https://www.gartner.com/doc/3084942/forecast-public-cloud-services-worldwide, 2015.

[51] E. von Glasersfeld. Learning as Constructive Activity. In *Proceedings of the 5th Annual Meeting of the North American Group of Psychology in Mathematics Education, Vol. 1. Montreal:*, 1983.

[52] E. von Glasersfeld. Questions and answers about radical constructivism. In M.K. Pearsall, editor, *Scope, sequence, and coordination of secondary school science*, pages 169–182. National Science Teachers Association, 1992.

[53] G. Goncalves, P. Endo, M. Santos, D. Sadok, J. Kelner, B. Melander, and J.-E. Mangs. Cloudml: An integrated language for resource, service and request description for d-clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 399–406, Nov 2011.

[54] David Mast Grant Thomson. Born On the Cloud. www-07.ibm.com/au/cloud2014/pdf/Born-On-The-Cloud.pdf, 2014.

[55] Sam Guinea, Gabor Kecskemeti, Annapaola Marconi, and Branimir Wetzstein. Multi-layered monitoring and adaptation. In *Proceedings of the 9th international conference on Service-Oriented Computing*, ICSOC'11, pages 359–373, Berlin, Heidelberg, 2011. Springer-Verlag.

[56] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 644–651, Washington, DC, USA, 2012. IEEE Computer Society.

[57] E. Holbæk-Hanssen, P. Håndlykken, and K. Nygaard. *"System Description and the Delta Language"*. Norsk Regnesentral, 1975.

[58] HP. The HP IT Service Management (ITSM) Reference Model. ftp://ftp.hp.com/pub/services/itsm/info/itsm_rmwp.pdf.

[59] IBM. Integrated service management and cloud computing:More than just technology best friends. https://www.ibm.com/ibm/files/E955200R99025N70/5Integrated_service_management_and_cloud_computing_644KB.pdf.

[60] IBM. WSLA Language Specification, version 1.0. http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf, 2001.

[61] IBM. IBM Cloud Computing Reference Architecture, 2013.

[62] Christian Inzinger, Stefan Nastic, Sanjin Sehic, Michael Vögler, Fei Li, and Schahram Dustdar. Madcat - a methodology for architecture and deployment of cloud application topologies. In *8th International Symposium on Service-Oriented System Engineering*. IEEE, 2014.

[63] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 58–65, May 2013.

[64] Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 658–665, Washington, DC, USA, 2011. IEEE Computer Society.

[65] Nima Kaviani, Eric Wohlstadter, and Rodger Lea. Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. In Gerti Kappel, Zakaria Maamar, and HamidR. Motahari-Nezhad, editors, *Service-Oriented Computing*,

volume 7084 of *Lecture Notes in Computer Science*, pages 157–171. Springer Berlin Heidelberg, 2011.

[66] Raman Kazhamiakin, Branimir Wetzstein, Dimka Karastoyanova, Marco Pistore, and Frank Leymann. Adaptation of service-based applications based on process quality factor analysis. In *Proceedings of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 395–404, Berlin, Heidelberg, 2009. Springer-Verlag.

[67] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. RodrÃŋguez, O. MercÃĺ, A.Cs. Marosi, J. Marco, and X. Franch. Enhancing Federated Cloud Management with an Integrated Service Monitoring Approach. *Journal of Grid Computing*, 11(4):699–720, 2013.

[68] Praveenkumar Kondikoppa, Chui-Hui Chiu, and Seung-Jong Park. MapReduce Performance in Federated Cloud Computing Environments. In Keesook J. Han, Baek-Young Choi, and Sejun Song, editors, *High Performance Cloud Auditing and Applications*, pages 301–322. Springer New York, 2014.

[69] Y. Kouki, F.A. de Oliveira, S. Dupont, and T. Ledoux. A language support for cloud elasticity management. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 206–215, May 2014.

[70] P. Kranas, V. Anagnostopoulos, A. Menychtas, and T. Varvarigou. ElaaS: An Innovative Elasticity as a Service Framework for Dynamic Management across the Cloud Stack Layers. In *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 1042 –1049, july 2012.

[71] Kyriakos Kritikos, Jörg Domaschka, and Alessandro Rossini. Srl: A scalability rule language for multi-cloud environments. In *CloudCom 2014: 6th IEEE International Conference on Cloud Computing Technology and Science*, pages 1–9. IEEE Computer Society, 2014.

[72] Katrina LaCurts, Jeffrey C. Mogul, Hari Balakrishnan, and Yoshio Turner. Cicada: Introducing predictive guarantees for cloud networks. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, June 2014. USENIX.

[73] Duc-Hung Le, Hong-Linh Truong, Georgiana Copil, Stefan Nastic, and Schahram Dustdar. Salsa: A framework for dynamic configuration of cloud services. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 146–153. IEEE, 2014.

[74] Philipp Leitner, Johannes Ferner, Waldemar Hummer, and Schahram Dustdar. Data-driven and automated prediction of service level agreement violations in service compositions. *Distributed and Parallel Databases*, 31(3):447–470, 2013.

[75] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. Cloudprophet: towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, New York, NY, USA, 2011. ACM.

[76] Ge Li, F. Pourraz, and P. Moreaux. Psla: A paas level sla description language. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 452–457, March 2014.

[77] Zheng Li, Liam O'Brien, He Zhang, and Rainbow Cai. On a catalogue of metrics for evaluating commercial cloud services. In *2012 ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pages 164 –173, sept. 2012.

[78] Rong Liu and Juhnyoung Lee. It incident management by analyzing incident relations. In Chengfei Liu, Heiko Ludwig, Farouk Toumani, and Qi Yu, editors, *Service-Oriented Computing*, volume 7636 of *Lecture Notes in Computer Science*, pages 631–638. Springer, 2012.

[79] M. Maciandas, J.O. Fito and, and J. Guitart. Rule-based sla management for revenue maximisation in cloud computing markets. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 354 –357, oct. 2010.

[80] Simon J. Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 131–140, New York, NY, USA, 2011. ACM.

[81] Ming Mao and M. Humphrey. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, pages 67–78, May 2013.

[82] KV Mardia, JT Kent, and JM Bibby. Multivariate analysis. *Probability and Mathematical Statistics, London: Academic Press, 1979*, 1, 1979.

[83] Stefano Mariani, Hong-Linh Truong, Georgiana Copil, Andrea Omicini, and Schahram Dustdar. Coordination-aware elasticity. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, pages 465–472. IEEE, 2014.

[84] S. Marston, Zhi Li, S. Bandyopadhyay, and A. Ghalsasi. Cloud computing - the business perspective. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1 –11, jan. 2011.

[85] P. Martin, A. Brown, W. Powley, and J. L. Vazquez-Poletti. Autonomic management of elastic services in the cloud. In *Proceedings of the 2011 IEEE Symposium on Computers and Communications*, ISCC '11, pages 135–140, Washington, DC, USA, 2011. IEEE Computer Society.

[86] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology (NIST)*, 2009.

[87] Richard A. Meyer and Love H. Seawright. A virtual machine time-sharing system. *IBM Systems Journal*, 9(3):199–218, 1970.

[88] M. Miglierina, G.P. Gibilisco, D. Ardagna, and E. Di Nitto. Model based control for multi-cloud applications. In *2013 5th International Workshop on Modeling in Software Engineering (MiSE)*, pages 37–43, May 2013.

[89] Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. Mela: Monitoring and analyzing elasticity of cloud services. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 80–87. IEEE, 2013.

[90] Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. MELA: Monitoring and Analyzing Elasticity of Cloud Services. In *2014 IEEE Sixth International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.

[91] Daniel Morán, Luis M. Vaquero, and FermÃŋn GalÃąn. Elastically ruling the cloud: Specifying application's behavior in federated clouds. In *IEEE CLOUD'11*, pages 89–96, 2011.

[92] V.I. Munteanu, A. Edmonds, T.M. Bohnert, and T.-F. Fortis. Cloud Incident Management, Challenges, Research Directions, and Architectural Approach. In *International Conference on Utility and Cloud Computing*, pages 786–791. IEEE/ACM, 2014.

[93] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas. Dependable horizontal scaling based on probabilistic model checking. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 31–40, May 2015.

[94] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Cloud elasticity using probabilistic model checking. *CoRR*, abs/1405.4699, 2014.

[95] Stefan Nastic, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. Governing elastic iot cloud systems under uncertainty. In *2015 IEEE Seventh International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015.

[96] National Institute of Standards and Technology. NIST Cloud Computing Reference Architecture. http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_SP_500-292_-_090611.pdf.

[97] Kristen Nygaard. Program development as a social activity. In *In H. -J. Kugler (Ed.), Proceedings of Information Processing 86*, pages 189–198. North-Holland, 1986.

[98] OASIS Committee. Topology and Orchestration Specification for Cloud Applications (TOSCA), Working Draft. https://www.oasisopen.org/committees/download.php/47871/tosca-primer-v1%200-wd04-rev01-clean.doc, January 2013.

[99] Oracle. Oracle Cloud Computing Reference Architecture. http://www.oracle.com/technetwork/topics/entarch/oracle-wp-cloud-ref-arch-1883533.pdf, 2012.

[100] Open Virtualization Format(OVF) Whitepaper. http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf.

[101] S.K. Panda and P.K. Jana. A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment. In *2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV)*, pages 82–87, Jan 2015.

[102] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier. A Federated Multi-cloud PaaS Infrastructure. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 392–399, June 2012.

[103] D.F. Parkhill. *The Challenge of the Computer Utility*. Number p. 246 in The Challenge of the Computer Utility. Addison-Wesley Publishing Company, 1966.

[104] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[105] Dana Petcu, Silviu Panica, Călin Şandru, Ciprian Dorin Crăciun, and Marian Neagul. Experiences in building an event-driven and deployable platform as a service. In *Web Information Systems Engineering-WISE 2012*, pages 666–672. Springer, 2012.

[106] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. QoS Guarantees and Service Differentiation for Dynamic Cloud Applications. *IEEE Transactions on Network and Service Management*, 10(1):43–55, March 2013.

[107] Mathias Sallé. It service management and it governance: review, comparative analysis and their impact on utility computing. *Hewlett-Packard Company*, pages 8–17, 2004.

[108] Americo Sampaio and Nabor Mendonça. Uni4Cloud: An Approach Based on Open Standards for Deployment and Management of Multi-cloud Applications. In

Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing, SECLOUD '11, pages 15–21, New York, NY, USA, 2011. ACM.

[109] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *2011 31st International Conference on Distributed Computing Systems (ICDCS)*, pages 559 –570, june 2011.

[110] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14. ACM, 2011.

[111] Thomas B Sheridan. *Telerobotics, automation, and human supervisory control*. MIT press, 1992.

[112] Thomas B Sheridan. Adaptive automation, level of automation, allocation authority, supervisory control, and adaptive control: Distinctions and modes of adaptation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(4):662–667, 2011.

[113] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, DTIC Document, 1978.

[114] Hong-Linh Truong Stefan Nastic. Infrastructure Level Uncertainties V2.0. Technical report, Distributed Systems Group, TU Wien, July 2015.

[115] Stefan Tai, Philipp Leitner, and Schahram Dustdar. Design by Units: Abstractions for Human and Compute Resources for Elastic Systems. *IEEE Internet Computing*, 16(4), 2012.

[116] Bing Tang and Mingdong Tang. Bayesian model-based prediction of service level agreement violations for cloud services. In *Theoretical Aspects of Software Engineering Conference (TASE), 2014*, pages 170–176, Sept 2014.

[117] Nikola Tankovic, T.G Grbac., Hong-Linh Truong, and Schahram Dustdar. Transforming vertical web applications into elastic cloud applications. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 135–144, March 2015.

[118] Rafael Tolosana-Calasanz, Jose Angel Banares, Congduc Pham, and Omer F. Rana. Resource management for bursty streams on multi-tenancy cloud environments. *Future Generation Computer Systems*, 2015.

[119] Demetris Trihinas, George Pallis, and Marios D. Dikaiakos. JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014.

[120] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris. Automated, elastic resource provisioning for nosql clusters using tiramola. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 34–41, 2013.

[121] Dimitrios Tsoumakos, Ioannis Konstantinou, Christina Boumpouka, Spyros Sioutas (Ionian University), and Nectarios Koziris. Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 34–41. IEEE Computer Society, 2013.

[122] Edward A Van Schaik. *A Management System for the Information Business: Organizational Analysis*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[123] Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino, Massimilano Rak, and Dana Petcu. A cloud agency for sla negotiation and management. In *Proceedings of the 2010 conference on Parallel processing*, Euro-Par 2010, pages 587–594, Berlin, Heidelberg, 2011. Springer-Verlag.

[124] Akshat Verma, Gautam Kumar, and Ricardo Koller. The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial Track*, pages 11–16, New York, NY, USA, 2010. ACM.

[125] Michael Vögler, Johannes M Schleicher, Christian Inzinger, Stefan Nastic, Sanjin Sehic, and Schahram Dustdar. Leonore–large-scale provisioning of resource-constrained iot deployments. In *9th International Symposium on Service-Oriented System Engineering*, pages 78–87. IEEE, 2014.

[126] Lizhe Wang, G. von Laszewski, Marcel Kunze, and Jie Tao. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, 04/2010 2010.

[127] Qingyang Wang, S. Malkowski, Y. Kanemasa, D. Jayasinghe, PengCheng Xiong, C. Pu, M. Kawaba, and L. Harada. The Impact of Soft Resource Allocation on n-Tier Application Scalability. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1034 –1045, may 2011.

[128] Wei Wang, Baochun Li, and Ben Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In *Presented as part of the 10th International Conference on Autonomic Computing*, pages 13–22, Berkeley, CA, 2013. USENIX.

[129] Jie Yang, Jie Qiu, and Ying Li. A Profile-Based Approach to Just-in-Time Scalability for Cloud Applications. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, CLOUD '09, pages 9–16, Washington, DC, USA, 2009. IEEE Computer Society.

[130] Li Yu and D. Thain. Resource management for elastic cloud workflows. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 775–780, 2012.

[131] Rostyslav Zabolotnyi, Philipp Leitner, Stefan Schulte, and Schahram Dustdar. SPEEDL - A declarative event-based language to define the scaling behavior of cloud applications. In *2015 IEEE World Congress on Services, SERVICES 2015, New York City, NY, USA, June 27 - July 2, 2015*, pages 71–78, 2015.

[132] Arkady B. Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing as a service and big data. *CoRR*, abs/1301.0159, 2013.

[133] HongJun Zhan and Wei Zhang. The Operation and Maintenance Management System of the Cloud Computing Data Center Based on ITIL. In *Advances in Computer Science and its Applications*, volume 279, pages 1103–1108. Springer, 2014.

[134] Li Zhang, Xiaoqiao Meng, Shicong Meng, and Jian Tan. K-scope: Online performance tracking for dynamic cloud applications. In *Presented as part of the 10th International Conference on Autonomic Computing*, pages 29–32, Berkeley, CA, 2013. USENIX.

Annex

Using rSYBL framework

rSYBL Deployment and Configuration

rSYBL exposes a RESTful API for interaction with the users, the lifecycle of an interaction being presented below, the service being produced by rSYBL Analysis component

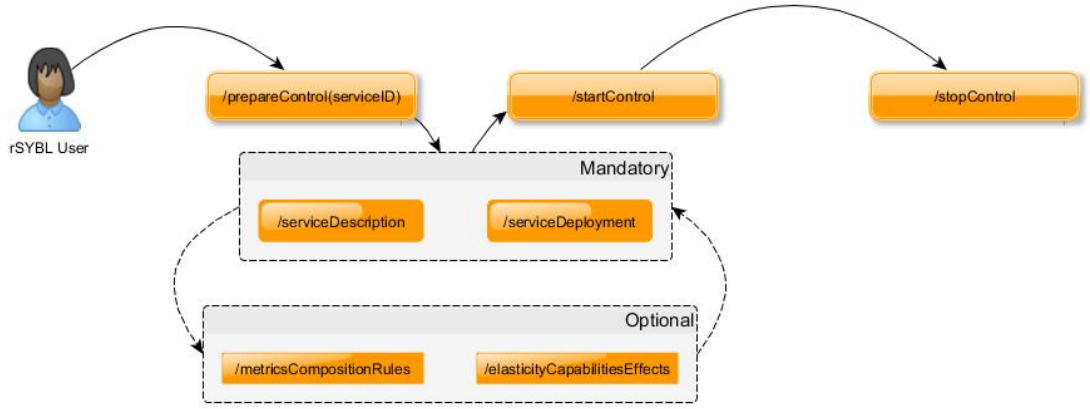


Figure 1: rSYBL initialization steps

Monitoring Plugin Configuration

1. **Ganglia Plugin**
   For working with simple ganglia plugin some information has to be specified:

   - The ganglia plugin class needs to be specified in the configuration file: in `config.properties` `MonitoringPlugin = at.ac.tuwien.dsg.sybl.monitorandenforcement .monitoringPlugins.gangliaMonitoring.MonitoringGangliaAPI`
   - Ganglia IP needs to be specified in the configuration file (config.properties)
   - Ganglia Port needs to be specified in the configuration file (config.properties)

2. MELA Plugin

The main plugin class implementing MonitoringInterface needs to be specified in the `config.properties` file:

```
MonitoringPlugin = at.ac.tuwien.dsg.rSybl.dataProcessingUnit.
    monitoringPlugins.melaPlugin.MELA_API and
MonitoringServiceURL = http://MELA_HOST_IP:MELA_HOST_PORT/MELA/
    REST_WS
```

The user should consult the configurations needed for running MELA[12].

Enforcement Plugin Configuration

Current version of rSYBL has the following plugins:

- OpenStack, for your OpenStack-based private cloud
- Flexiant, for controlling your application with rSYBL on the Flexiant[9] cloud, using FCO[10] for controlling resources.
- DryRun, for testing purposes. It outputs the actions rSYBL would like to enforce, without further enforcement.

The OpenStack enforcement plugin needs to be specified in `config.properties`:

```
EnforcementPlugin =  at.ac.tuwien.dsg.rSybl.
    cloudInteractionUnit.enforcementPlugins.openstack.
    EnforcementOpenstackAPI
CertificateName= e.g., mycert - The name of the certificate
    used
CertificatePath=e.g., config/mycert.pem - The path towards the
    certificate
CloudAPIType = e.g., openstack-nova -  Type of the cloud
    middleware used
CloudAPIEndpoint= e.g., in this case DSG local cloud     http
    ://openstack.infosys.tuwien.ac.at:5000/v2.0 - Endpoint for
    accessing the cloud infrastructure
CloudUser= e.g., georgiana.copil - Username for accessing cloud
     infrastructure
CloudPassword= e.g., mypassword - Password for accessing cloud
    infrastructure
```

The Flexiant enforcement plugin needs to be specified in `config.properties`:

[9] www.flexiant.com

[10] http://www.flexiant.com/flexiant-cloud-orchestrator/

```
EnforcementPlugin = at.ac.tuwien.dsg.rSybl.
    cloudInteractionUnit.enforcementPlugins.flexiant.
    EnforcementFlexiantAPI
UserEmailAddress = x@domain.com - Email address of Flexiant
    user
ApiUserName=x@domain.com - User name of Flexiant user (usually
    matches e-mail address)
CustomerUUID =XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - customer
    UUID on the FCO platform, can be obtained from user
    interface -> settings
Password =password - your password on Flexiant cloud
ENDPOINT_ADDRESS_PROPERTY=https://api.sd1.flexiant.net:4442 -
    current endpoint exposed by Flexiant
VdcUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - the virtual
    datacenter where your application will be deployed &
    controlled
DefaultProductOfferUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX -
    the default type of VM which you would like to start (e.g.,
    886ae014-0613-3cc8-a790-16251471e624)
DeploymentInstanceUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX -
    the deployment instance to which the resources will be added
ClusterUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - the current
    cluster where the application is deployed
NetworkUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - the network
    which the application is using
SSHKey=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - the UUID of the
    ssh key with which the VMs need to start (in case one needs
    passwordless access to them)
```

The DryRun enforcement plugin, used for testing purposes, needs to be specified in `config.properties`

```
EnforcementPlugin = at.ac.tuwien.dsg.rSybl.
    cloudInteractionUnit.enforcementPlugins.dryRun.
    DryRunEnforcementAPI
```

Application Specific Configurations

The application-specific configuration can be sent to the rSYBL in three ways, depending on the information structure.

rSYBL bootstrapping For these configurations, rSYBL needs to be deployed and started (see steps below)

For configuring rSYBL using its predefined, simple XML descriptions, the rSYBL service exposes the following web methods to be called in this order:

- `prepareControl(cloudServiceID)` (PUT method, accepting xml/application)- prepare cloudServiceID cloud service for control

- `serviceDescription(descriptionInfo)` (PUT method, accepting xml/application). Example of such a description is available at https://raw.githubusercontent.com/tuwiendsg/rSYBL/master/rSYBL-control-service-pom/rSYBL-analysis-engine/src/main/resources/config/serviceDescription.xml and is described on the rSYBL webpage[11].

- `serviceDeployment(deploymentInfo)` (PUT method, accepting xml/application). Example of such a description is available at https://github.com/tuwiendsg/rSYBL/blob/master/rSYBL-control-service-pom/ rSYBL-analysis-engine/src/main/resources/config/newDeploymentDescription.xml and is described on the rSYBL webpage[11].

- *Optional*: `metricsCompositionRules(metricsCompositionRules)` (PUT method, accepting xml/application). Example of such a description is available at https://github.com/tuwiendsg/rSYBL/blob/master/rSYBL-control-service-pom/rSYBL-analysis-engine/src/main/resources/config/compositionRules.xml and is described on the MELA webpage[12].

- *Optional*: `elasticityCapabilitiesEffects(elasticityCapabilities Effects)` (PUT method, accepting json/application). Example of such a description is available at https://github.com/tuwiendsg/rSYBL/blob/master/rSYBL-control-service-pom/rSYBL-analysis-engine/src/main/resources/config/effects.json.

- `startControl(cloudServiceID)` (PUT method, accepting xml/application)- start cloudServiceID cloud service control

- When done, gracefully shut down rSYBL control service: `stopControl (cloudServiceID)` (PUT method, accepting xml/application)- stop cloudServiceID cloud service control

- During runtime, replace existing requirements: `replaceRequirements (cloudServiceID,cloudServiceRequirements)` (PUT method, accepting xml/application)- replace existing requirements with others cloudServiceID cloud service control

Application description rSYBL accepts the applications described both using the rSYBL specific description, and using TOSCA. One such example is shown below, describing the SCAN cancer research application[13]:

[11] http://www.infosys.tuwien.ac.at/research/viecom/SYBL
[12] `http://tuwiendsg.github.io/MELA`
[13] http://www.celarcloud.eu/newsroom/scan-celar-and-getting-the-most-out-of-your-hybrid-cloud

```
<?xml version="1.0" encoding="UTF-8"?>
<tosca:Definitions xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xmlns:elasticity="http://www.
    example.org/NewXMLSchema" xmlns:sybl="http://www.
    example.org/SYBL" xmlns:tosca="http://docs.oasis-
    open.org/tosca/ns/2011/12" id="hi">
<tosca:ServiceTemplate xsi:type="elasticity:
    TServiceTemplateExtension" id="hello" name="
    SCANOKLaptop">
<tosca:BoundaryDefinitions xsi:type="elasticity:
    TBoundaryDefinitionsExtension">
<tosca:Properties>
<elasticity:ServiceProperties>
<elasticity:Version>1.0</elasticity:Version>
</elasticity:ServiceProperties>
</tosca:Properties>
</tosca:BoundaryDefinitions>
<tosca:TopologyTemplate>
<tosca:NodeTemplate xsi:type="elasticity:
    TNodeTemplateExtension" id="C785287430" maxInstances
    ="1" minInstances="1" name="scheduler" x="35" y
    ="55">
<tosca:Properties>
<elasticity:NodeProperties>
<elasticity:Flavor>vcpus:4 ram:4096 disk:40</elasticity
    :Flavor>
</elasticity:NodeProperties>
</tosca:Properties>
<tosca:Policies>
<tosca:Policy name="CONSTRAINT queueLength$&amp;lt;3"
    policyRef="C19330703800" policyType="sybl:Constraint
    "/>
<tosca:Policy name="CONSTRAINT workerUtilisation$&amp;
    gt;0.75" policyRef="C19330703801" policyType="sybl:
    Constraint"/>
</tosca:Policies>
<tosca:DeploymentArtifacts>
<tosca:DeploymentArtifact artifactRef="5b1ac218-1cdf
    -4286-8000-09991bd07752" artifactType="elasticity:
    ImageArtifactPropertiesType" name="5b1ac218-1cdf
    -4286-8000-09991bd07752"/>
</tosca:DeploymentArtifacts>
</tosca:NodeTemplate>
```

```
<tosca:NodeTemplate xsi:type="elasticity:
    TNodeTemplateExtension" id="C492638789" maxInstances
    ="100" minInstances="1" name="worker" x="205" y
    ="55">
<tosca:Properties>
<elasticity:NodeProperties>
<elasticity:Flavor>vcpus:4 ram:4096 disk:40</elasticity
    :Flavor>
</elasticity:NodeProperties>
</tosca:Properties>
<tosca:Policies>
<tosca:Policy name="CONSTRAINT freeDiskProp$&amp;gt
    ;0.25" policyRef="C17803292480" policyType="sybl:
    Constraint"/>
<tosca:Policy name="CONSTRAINT freeDiskProp$&amp;lt
    ;0.75" policyRef="C17803292481" policyType="sybl:
    Constraint"/>
<tosca:Policy name="STRATEGY CASE violated(C19330703801
    ) : scaleIn(worker)" policyRef="C17803292482"
    policyType="sybl:Strategy"/>
<tosca:Policy name="STRATEGY CASE violated(C19330703800
    ) : scaleOut(worker)" policyRef="C17803292483"
    policyType="sybl:Strategy"/>
<tosca:Policy name="STRATEGY CASE violated(C17803292480
    ) : attachDisk(worker)" policyRef="C17803292484"
    policyType="sybl:Strategy"/>
<tosca:Policy name="STRATEGY CASE violated(C17803292481
    ) : detachDisk(worker)" policyRef="C17803292485"
    policyType="sybl:Strategy"/>
</tosca:Policies>
<tosca:DeploymentArtifacts>
<tosca:DeploymentArtifact artifactRef="5b1ac218-1cdf
    -4286-8000-09991bd07752" artifactType="elasticity:
    ImageArtifactPropertiesType" name="5b1ac218-1cdf
    -4286-8000-09991bd07752"/>
</tosca:DeploymentArtifacts>
</tosca:NodeTemplate>
</tosca:TopologyTemplate>
</tosca:ServiceTemplate>
<tosca:ArtifactTemplate id="init_gatk_worker.sh" type="
    elasticity:ScriptArtifactPropertiesType" name="
    SDinit_gatk_worker.sh">
<tosca:Properties>
```

```
<elasticity:ScriptArtifactProperties>
<elasticity:Language>Shell</elasticity:Language>
</elasticity:ScriptArtifactProperties>
</tosca:Properties>
<tosca:ArtifactReferences>
<tosca:ArtifactReference reference="Scripts/
    init_gatk_worker.sh"/>
</tosca:ArtifactReferences>
</tosca:ArtifactTemplate>
<tosca:NodeTypeImplementation name="name" nodeType="
    worker">
<tosca:ImplementationArtifacts>
<tosca:ImplementationArtifact artifactRef="
    init_gatk_worker.sh" artifactType="elasticity:
    ScriptArtifactPropertiesType" interfaceName="
    Lifecycle" operationName="execute"/>
<tosca:ImplementationArtifact artifactRef="
    handle_prescale.sh" artifactType="ScriptArtifact"
    interfaceName="PreScale" operationName="STRATEGY
    scaleIn (worker)"/>
<tosca:ImplementationArtifact artifactRef="
    handle_prescale.sh" artifactType="ScriptArtifact"
    interfaceName="PreScale" operationName="STRATEGY
    scaleIn (worker)"/>
<tosca:ImplementationArtifact artifactRef="
    handle_postscale.sh" artifactType="ScriptArtifact"
    interfaceName="PostScale" operationName="STRATEGY
    attachDisk (worker)"/>
<tosca:ImplementationArtifact artifactRef="
    handle_prescale.sh" artifactType="ScriptArtifact"
    interfaceName="PreScale" operationName="STRATEGY
    detachDisk (worker)"/>
</tosca:ImplementationArtifacts>
</tosca:NodeTypeImplementation>
<tosca:ArtifactTemplate id="init_scheduler.sh" type="
    elasticity:ScriptArtifactPropertiesType" name="
    SDinit_scheduler.sh">
...
</tosca:Definitions>
```

rSYBL API

Method	Type	Input	Output	Description
/id/prepareControl	PUT	application/xml	-	Sets the rSYBL controller in "prepare mode", for receiving all the necessary information for control.
/id/description/tosca	PUT	application/xml	-	Sends to rSYBL the TOSCA description of service with ID id
/id/description	PUT	application/xml	-	Sets rSYBL specific application structural description (e.g., here) for service with ID id. When this is sent TOSCA description is not needed.
/id/deployment	PUT	application/xml	-	Sets current service deployment description for service with ID id (e.g., here).
/id/elasticityCapabilities Effects	PUT	application/json	-	Sets effects expected for primitive operations in JSON format (e.g., here) for service with ID id.
/id/compositionRules	PUT	application/xml	-	Sets composition rules for service metrics in MELA-specific XML format (e.g., here) for service with ID id.
/id/startControl	PUT	application/xml	-	Starts the control for the service with ID id.
/id/startControlOnExis ting	PUT	application/xml	-	Starts the control on the service with ID id, which is already deployed and monitored.
/id/stopControl	PUT	application/xml	-	Stops the control (equivalent with a pause in control, for when the service is manually modified).
/id	DELETE	-	-	De-registers from rSYBL and MELA, and undeploys from the cloud the service with ID id.
/managedService/id	DELETE	-	-	De-registers from rSYBL and MELA service with ID id (does NOT undeploy) from the cloud.
/id/onDemandControl/ unhealthy	PUT	plain/txt	-	Triggers health fix (recursive restart in top-down mode) for service with ID id, for service part sent as parameter (in the method body).

Continued on next page

rSYBL API – *Continued from previous page*

Method	Type	Input	Output	Description
/id/startTEST	PUT	application/xml	-	Starts the rSYBL controller in testing mode
/id/servicePartID/ testElasticityCapability/capabilityID	PUT	application/xml	-	Enforces, for service with ID id, capability with capabilityID for service part with servicePartID.
/id/servicePartID/ testElasticityCapability/pluginID/capabilityID	PUT	application/xml	-	Enforces, for service with ID id, capability with capabilityID for service part with servicePartID, by using a specific enforcement plugin of rSYBL, with ID pluginID
. /id/description	GET	-	application/xml	Gets current application structural description used, in rSYBL specific format (e.g., here)
/id/elasticityRequirements/xml	GET	-	application/xml	Returns elasticity requirements for the service with ID id, in XML format.
/id/elasticityRequirements/plain	GET	-	text/plain	Returns elasticity requirements for the service with ID id, in SYBL format.
/id/structuralData/json	GET	-	application/json	Returns the service structural description in JSON format.
/elasticservices	GET	-	text/plain	Returns the ids of all currently controlled services, separated by ','.
/id/deployment	POST	application/xml	-	Modifies current service deployment description for service with ID id (e.g., here).
/id/description	POST	application/xml	-	Modifies for service with ID id, the service description which is sent in the body of this call.
/id/compositionRules	POST	application/xml	-	Modifies composition rules for service with ID id (i.e., old ones are being replaced with the new ones sent in the message body).

Continued on next page

rSYBL API – *Continued from previous page*

Method	Type	Input	Output	Description
/id/elasticityRequirements/xml	POST	application/xml	-	Modifies elasticity requirements for the service with ID id, with the ones received. The received requirements are for this method in XML format.
/id/elasticityCapabilitiesEffects	POST	application/json	-	Modifies elasticity capabilities effects for service with ID id, with the ones sent in JSON format.
/id/replaceRequirements/plain	POST	text/plain	-	Replaces elasticity requirements for service with ID id with the ones received in SYBL format.

Deployment

For deploying and starting rSYBL, there are two options:

- Deploy the war artifact generated by the rSYBL-analysis-engine component in an existing Web Server (e.g., Tomcat)
- Start the executable war artifact which also contains the web server
- Run rSYBL-service which can be found on the root of rSYBL repo (this works in Linux-based systems)

Note: In case Java version is less than 7, the following jvm options are needed:
`-Djsse.enableSNIExtension=false -Djavax.xml.bind.JAXBContext = com.sun.xml.internal.bind.v2.ContextFactory`

Multiple enforcement mechanisms – Multi-cloud control configuration

rSYBL can control services which have different service parts (e.g., service topologies/-composite components) deployed on different cloud providers. Moreover, rSYBL can also use user-defined control, which can be defined following the steps in the next section, in order to control cloud services.

An example of a client which follows this entire lifecycle, with two clouds, as presented in the figure below, is available at https://github.com/tuwiendsg/rSYBL/tree/master/rSYBL-control-service-pom/rSYBL-client/rSYBL%20Python%20clients/multipleEnforcementMechanisms_multiCloud/lifecycle.py. The description of the cloud service together with its requirements stays the same, for the case the cloud service is deployed on two clouds with similar functionalities.

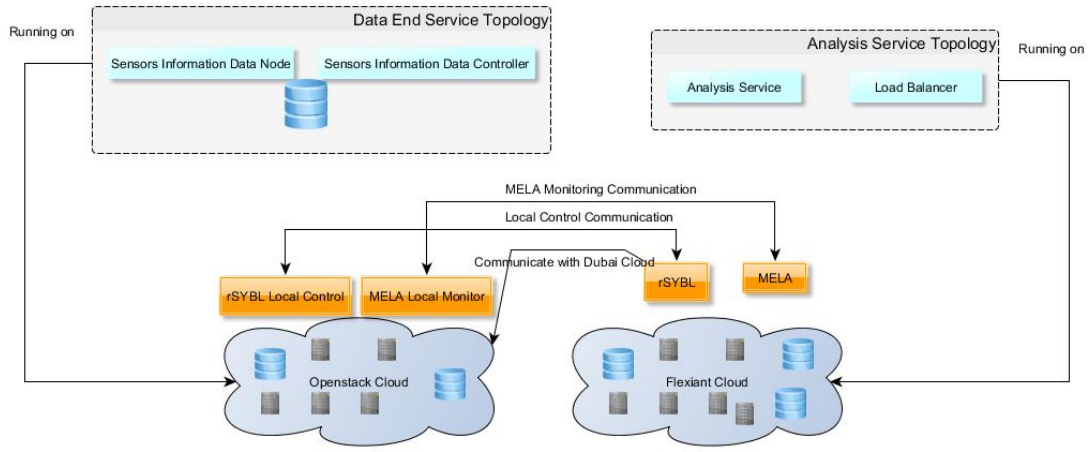


Figure 2: Multi-cloud example

The configuration of enforcement mechanisms for the case above is[14]

```
MultipleEnforcementPlugins = flexiant:at.ac.tuwien.dsg.rSybl.
    cloudInteractionUnit.enforcementPlugins.flexiant.
    EnforcementFlexiantAPI, openstack:at.ac.tuwien.dsg.rSybl.
    cloudInteractionUnit.enforcementPlugins.openstack.
    EnforcementOpenstackAPI
```

This way, we say that we will refer to the EnforcementFlexiantAPI as flexiant, and to EnforcementOpenstackAPI as openstack, and we will use them both.

However, there are some differences in terms of effects description and deployment description. The single difference, next to adding extra plugins in config.properties, is that when specifying capabilities of different nodes, one needs to specify which enforcement mechanism comes from which plugin:

```
<DeploymentUnit serviceUnitID="EventProcessingServiceUnit"
defaultFlavor="m1.tiny" defaultImage="d279e72d-4bba-3c7f-8330-9
    ad4d29e8dfe">
<AssociatedVM IP="109.231.122.250" UUID="131627fe-d5f3-3518-8
    bf5-7d16a932df8c"/>
<AssociatedVM IP="109.231.122.251" UUID="9c99d99b-b2c4-30ee-
    a87f-67e7370291ad"/>
<ElasticityCapability Name="flexiant.scaleIn"/>
<ElasticityCapability Name="flexiant.scaleOut"/>
</DeploymentUnit>

<DeploymentUnit serviceUnitID="DataNodeServiceUnit"
defaultFlavor="m1.tiny" defaultImage="728a8bfc-2af7-4e8c-a782
    -05e292deef81" >
<AssociatedVM IP="10.99.0.91"/>
<ElasticityCapability Name="openstack.scaleIn"/>
<ElasticityCapability Name="openstack.scaleOut"/>
</DeploymentUnit>
```

The description above is normally introduced not by the user, but by the deployment/-configuration framework (e.g., SALSA) as the deployment should be done automatically.

Extending rSYBL

For customizing rSYBL in what Monitoring and Enforcement is concerned, the following steps need to be followed.

For creating new plugins, an API of the plugin needs to implement the MonitoringInterface and respectively EnforcementInterface. The basic metrics and actions appear by name, while the ones which are specific to the applications/plugins are to be interfaced

[14]Example of multi-cloud config file https://github.com/tuwiendsg/rSYBL/tree/master/starting%20rSYBL/rSYBL%20Python%20clients/config_multiCloud.properties

through getMetric and enforceAction actions. Moreover, the needed configuration data needs to be added and processed from the config file of the Analysis Engine, which is the main module of rSYBL tool.

After adding new plugins, the rSYBL tool needs to be recompiled and re-deployed, and these plugins need to be specified in `config.properties` and associated, in the case of the enforcement plugin, in the deployment description, with the node (e.g., service unit, service topology) which exposes this implemented capability.

GEORGIANA COPIL

PERSONAL INFORMATION

Born in Romania, 23 April 1987

email e.copil@dsg.tuwien.ac.at

website http://dsg.tuwien.ac.at/staff/ecopil/

WORK EXPERIENCE

2014 - present University Assistant, Distributed Systems Group, TU Wien — Vienna, Austria

Research activities on elasticity control in the cloud, and teaching activities.

2012- 2014 Project Assistant, Distributed Systems Group, TU Wien — Vienna, Austria

Research activities on elasticity control in the cloud.

2011 Summer Intern, IBM Research Haifa — Tel Aviv site, Israel

Research activities on cloud storage for energy efficiency and performance improvement.

2010–2012 Research Assistant, Distributed Systems Research Laboratory — Cluj-Napoca, Romania

Research and development for GAMES[1] (Green Active Management of Energy in IT Service centers) FP7 EU Project.

2008–2010 Software Developer Intern, National Instruments Corporation — Cluj-Napoca, Romania

Developing features for LabView, the development environment for a visual programming language from National Instruments.

EDUCATION

2012-2016 Technische Universität Wien

PhD in Computer Science (title: Dr.)

Description: The dissertation thesis is prepared under the supervision of Prof. Univ. Dr. Schahram, having the title *Cloud Services Elasticity Control: from requirements specification to operations management*

2010-2012 Technical University of Cluj-Napoca

MSc. in Artificial Intelligence and Computer Vision (title: Dipl. Ing.)

Description: The dissertation thesis was prepared under the supervision of Professor Ioan Salomie, having the title *Targeting Service Level Agreements in Green Cloud Computing: Management and Negotiation*

2006-2010 Technical University of Cluj-Napoca

Bachelor of Computer Science (title: Ing.)

Description: The diploma thesis was prepared under the supervision of professor Ioan Salomie, having the title *A self-adaptive model for managing energy efficiency in service centres: Adaptive Action Decision and Enforcement*

1 http://green-datacenters.eu

FULL PUBLICATION LIST

Journals

ACM Transactions on Internet Technology

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "rSYBL: a Framework for Specifying and Controlling Cloud Services Elasticity", ACM Transactions on Internet Technology (TOIT), 2015.

International Journal of Cooperative Information Systems

Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, Marios D. Dikaiakos, "Evaluating cloud service elasticity behavior", International Journal of Cooperative Information Systems

International Journal of Big Data Intelligence

Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "MELA: Elasticity Analytics for Cloud Services", International Journal of Big Data Intelligence

Ubiquitous Computing And Communication Journal

Tudor Cioara, Ionut Anghel, Ioan Salomie, Georgiana Copil, Daniel Moldovan,Barbara Pernici - "A Context Aware Self-Adapting Algorithm for Managing the Energy Efficiency of IT Service Centres" - Ubiquitous Computing And Communication Journal - Special Issue of RoEduNet, pp. 619 - 630, 2011.

Journal of Scalable Computing: Practice and Experience

Cristina Bianca Pop, Viorica Rozina Chifu, Ioan Salomie, Ramona Bianca Baico, Mihaela Dinsoreanu, Georgiana Copil - "A Hybrid Firefly-inspired Approach for Optimal Semantic Web Service Composition" - Journal of Scalable Computing: Practice and Experience,Volume 12, Number 3, 2011.

Conferences

2015

Stefan Nastic, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, Governing Elastic IoT Cloud Systems under Uncertainty, 7'th International Conference on Cloud Computing, CloudCom. Vancouver, 2015

Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "Supporting Cloud Service Operation Management for Elasticity", the 13th International Conference on Service Oriented Computing. Goa, India, 16-19 November, 2015

Tien-Dung Nguyen, Hong-Linh Truong, Georgiana Copil, Duc-Hung Le, Daniel Moldovan, Schahram Dustdar, "On Developing and Operating of Data Elasticity Management Process", (c)Springer-Verlag,13th International Conference on Service Oriented Computing (ICSOC 2015), Nov 16-19, 2015. Goa, India

Chris Smowton, Georgiana Copil, Hong-Linh Truong, Crispin Miller and Wei Xing, "Genome Analysis in a Dynamically Scaled Hybrid Cloud", The 11th IEEE International Conference on eScience, Munich, Germany, 31st August - 4th September 2015.

Hong-Linh Truong, Georgiana Copil, Schahram Dustdar, Duc-Hung Le, Daniel Moldovan, Stefan Nastic, "iCOMOT - Toolset for Managing IoT Cloud Systems", 16th IEEE International Conference on Mobile Data Management, 15-18 June, 2015, Pittsburg, USA. (Demo)

2014

Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "On Analyzing Elasticity Relationships of Cloud Services", 6'th International Conference on Cloud Computing, CloudCom. Singapore, 2014.

Duc-Hung Le, Hong-Linh Truong, Georgiana Copil, Stefan Nastic, Schahram Dustdar, "SALSA: a Framework for Dynamic Configuration of Cloud Services", 6'th International Conference on Cloud Computing, CloudCom. Singapore, 2014.

Stefano Mariani, Hong-Linh Truong, Georgiana Copil, Andrea Omicini, Schahram Dustdar, "Coordination-aware Elasticity", 7th IEEE/ACM International Conference on Utility and Cloud Computing, 8-11 December, London, 2014.

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "On Controlling Cloud Services Elasticity in Heterogeneous Clouds", 6th Cloud Control Workshop, 7th IEEE/ACM International Conference on Utility and Cloud Computing, 8-11 December, London, 2014.

Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, Marios Dikaiakos. "ADVISE - a Framework for Evaluating Cloud Service Elasticity Behavior" the 12th International Conference on Service Oriented Computing. Paris, France, 3-6 November, 2014. **(best paper award)**

Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "QUELLE - a Framework for Accelerating the Development of Elastic System s", Third European Conference on Service-Oriented and Cloud Computing - ESOCC 2014, 2-4 September, Manchester, United Kingdom.

Hong-Linh Truong, Schahram Dustdar, Georgiana Copil, Alessio Gambi, Waldemar Hummer, Duc-Hung Le, Daniel Moldovan, "CoMoT A Platform-as-a-Service for Elasticity in the Cloud", IEEE International Workshop on the Future of PaaS, IEEE International Conference on Cloud Engineering (IC2E 2014), Boston, Massachusetts, USA, 10-14 March 2014

2013

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "SYBL: an Extensible Language for Controlling Elasticity in Cloud Applications", 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 14-16, 2013, Delft, the Netherlands.

Alessio Gambi, Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "On Estimating Actuation Delays in Elastic Computing Systems", International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), May 20-21, 2013, San Francisco, USA.

Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar, "MELA: Monitoring and Analyzing Elasticity of Cloud Services", 5'th International Conference on Cloud Computing, CloudCom. Bristol, UK, 2-5 December, 2013

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "Multi-Level Elasticity Control of Cloud Services", the 11th International Conference on Service Oriented Computing. Berlin, Germany, on 2-5 December, 2013.

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar, "SYBL+MELA: Specifying, Monitoring, and Controlling Elasticity of Cloud Services", the 11th International Conference on Service Oriented Computing. Berlin, Germany, on 2-5 December, 2013.

2012

Georgiana Copil, Daniel Moldovan, Ioan Salomie, Tudor Cioara, Ionut Anghel, Diana Borza, "Cloud SLA negotiation for energy savingA particle swarm optimization approach", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2012.

Daniel Moldovan, Georgiana Copil, Ioan Salomie, Ionut Anghel, Tudor Cioara, "A membrane computing inspired packing solution and its application to service center workload distribution", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2012.

2011

Tudor Cioara, Ionut Anghel, Ioan Salomie, Georgiana Copil, Daniel Moldovan, Barbara Pernici., "A context aware self-adapting algorithm for managing the energy efficiency of it service centres," in Ubiquitous Computing and Communication Journal, 2010 9th , vol., no., pp.374-379, 24-26 June 2011

Tudor Cioara, Ionut Anghel, Ioan Salomie, Georgiana Copil, Daniel Moldovan, Alexander Kipp, "Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning", 10th International Symposium on Parallel and Distributed Computing (ISPDC), 2011.

Ionut Anghel, Tudor Cioara, Ioan Salomie, Georgiana Copil, Daniel Moldovan, Cristina Pop, "Dynamic frequency scaling algorithms for improving the CPU's energy efficiency", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2011.

Georgiana Copil, Tudor Cioara, Ionut Anghel, Ioan Salomie, Daniel Moldovan, Diana Borza, "A genetic-inspired negotiation algorithm for QoS and energy consumption tradeoffs in virtualized service centers", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2011.

Tudor Cioara, Ionut Anghel, Ioan Salomie, Georgiana Copil, Daniel Moldovan, Marius Grindean, "Time series based dynamic frequency scaling solution for optimizing the CPU energy consumption", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2011.

Tudor Cioara, Ionut Anghel, Ioan Salomie, Daniel Moldovan, Georgiana Copil, Pierluigi Plebani, "Dynamic consolidation methodology for optimizing the energy consumption in large virtualized service centers", Federated Conference on Computer Science and Information Systems (FedCSIS), 2011.

Cristina Bianca Pop, Viorica Rozina Chifu, Ioan Salomie, Ramona Bianca Baico, Mihaela Dinsoreanu, Georgiana Copil, "A hybrid firefly-inspired approach for optimal Semantic Web service composition", Scalable Computing: Practice and Experience, 2011.

2010

Tudor Cioara, Ionut Anghel, Ioan Salomie, Mihaela Dinsoreanu, Georgiana Copil, and Daniel Moldovan. "A reinforcement learning based self-healing algorithm for managing context adaptation". In Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services (iiWAS '10). ACM, New York, NY, USA, 859-862. DOI=10.1145/1967486.1967634 http://doi.acm.org/10.1145/1967486.1967634

Ioan Salomie,Tudor Cioara, Ionut Anghel, Daniel Moldovan, Georgiana Copil, Pierluigi Pleibani, "An Energy Aware Context Model for Green IT Service Centers", ICSOC 2010 International Workshops, PAASC, WESOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers.

Ionut Anghel, Tudor Cioara, Ioan Salomie, Mihaela Dinsoreanu, Georgiana Copil, Daniel Moldovan - "An Autonomic Context Management Model based on Machine Learning" - Proceedings of the 12th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 335 - 338, 2010.

Tudor Cioara, Ionut Anghel, Ioan Salomie, Mihaela Dinsoreanu, Georgiana Copil, and Daniel Moldovan, "A self-adapting algorithm for context aware systems," in Roedunet International Conference (RoEduNet), 2010 9th , pp.374-379, 24-26 June 2010

Book Chapters

Georgiana Copil, Daniel Moldovan, Duc-Hung Le, Hong-Linh Truong, Schahram Dustdar, Chrystalla Sofokleous, Nicholas Loulloudes, Demetris Trihinas, George Pallis, Marios D. Dikaiakos, Craig Sheridan, Evangelos Floros, Christos KK Loverdos, Kam Star, Wei Xing, On Controlling Elasticity of Cloud Applications in CELAR, Emerging Research in Cloud Distributed Computing Systems, Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

OTHER INFORMATION

Awards

2014 · ICSOC 2014 Best Paper Award for *ADVISE: A framework for evaluating cloud service elasticity behavior*
2011 · ISPDC 2011 Best Paper Award for *Energy Aware Dynamic Resource Consolidation Algorithm for Virtualized Service Centers based on Reinforcement Learning*

Languages

ROMANIAN · Mothertongue

Languages

ENGLISH · Advanced
DEUTSCH · Basic (simple words and phrases only)
FRENCH · Basic (simple words and phrases only)

December 3, 2015