UNIVERSITY OF CRETE
FACULTY OF SCIENCES AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

Cross-Layer Monitoring and Adaptation of Multi-Cloud Service-Based Applications

by

Chrysostomos Zeginis

Dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Heraklion, October 2014

UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE

Cross-Layer Monitoring and Adaptation of Multi-Cloud Service-Based Applications

Dissertation submitted by

Chrysostomos Zeginis

in partial fulfillment of the requirements for the
degree of Doctor of Philosophy in Computer Science

Author:	Chrysostomos Zeginis, University of Crete
Examination Committee:	Dimitris Plexousakis, Professor, University of Crete
	Christos Nikolaou, Professor, University of Crete
	Kostas Magoutis, Assistant Professor, University of Ioannina
	Evangelos Markatos, Professor, University of Crete
	Antonio Brogi, Professor, University of Pisa
	Schahram Dustdar, Professor, Vienna University of Technology
	Nikos Parlavantzas, Assistant Professor, Institut National des Sciences Appliquées
Department approval:	Panagiotis Tsakalides, Professor and Department Chair

Heraklion, October 2014

Abstract

Service-Oriented Architecture (SOA) emerged in the late 90s, introducing Web services as a new means for delivering software over a network. Nowadays, it provides many opportunities for businesses to automate their processes, by providing services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces. The success of many Web service related projects has shown that existing technologies enables implementing a true SOA. However, the evolution of Web services indicates that they are moving beyond the simple exchange of information to the concept of combining existing and new applications in order to provide more complex Service-based Applications (SBAs). Consequently, businesses will be able to create customizable composite SBAs, also integrating back-end and older technology systems found in local or remote applications.

With the advent of the new century, a number of additional pieces of the computing "puzzle" fell into place to complement SOA and reflect the new trends introduced by the *Internet of things (IoT)*, which refers to the interconnection of uniquely identifiable embedded computing devices with the existing Internet infrastructure. Cloud computing has emerged as a new paradigm for delivering "as-a-service" offerings to the end users: (i) *Software-as-a-Service (SaaS)* is the software delivery model adopted in Cloud computing, where Web services are made available to users on demand via the Internet from a Cloud provider, (ii) *Platform-as-a-Service (PaaS)* provides the platform to the application owners to deploy their applications on, and (iii) *Infrastructure-as-a-Service (IaaS)* is a resource provision-

ing model allowing Cloud providers to outsource equipment, i.e. *Virtual Machines (VMs)*, required by SaaS and PaaS.

The dynamic nature of Web services and the vulnerable execution environment in which they perform requires that their functional and non-functional (Quality of Service (QoS)) characteristics should be monitored. The monitoring process is essential in order to gain a clear view of how they perform within their operational environment, take management decisions and perform adaptation actions to modify and adjust their behavior, according to the new posed requirements. This dissertation addresses monitoring and adaptation of SBAs deployed on multiple Clouds, introducing an **E**vent based **C**ross-layer **M**onitoring and **A**daptation **F**ramework, named *ECMAF*. As SOA and Cloud architecture comprises a number of functional layers, including various application components, spanning from abstract business processes to concrete infrastructure resources, monitoring and adaptation should take into account the layers' dependencies in order to efficiently correlate the monitoring events and promptly determine the most suitable adaptation actions that should be triggered. The proposed approach takes into consideration all the Cloud and SOA layers comprising a multi-Cloud SBA.

To investigate the applicability of ECMAF and demonstrate its benefits, we have implemented and deployed a traffic management application on a multi-Cloud setup. This application has been suitably designed to optimally perform on multiple Clouds offering different storage and computational power. The Clouds used for the SBA's deployment exhibit a number of dependencies among the involved components across all the SOA and Cloud layers, captured in a component meta-model, which has been especially designed to model a snapshot of the current SBA deployment status and the dependencies of the active components. These dependencies are exploited by the ECMAF framework in order to extract valid event patterns causing specific Service-level Objectives (SLO) violations, which are further processed to form more complex ones, mapping to suitable adaptation strategies. Moreover, these dependencies are also enriched

at run-time to reflect new behaviors of the SBA, dictated by a context or an individual component's status modification. Additional meta-models for validating the monitoring events, as well as the adaptation strategies supported by the ECMAF's incorporated adaptation engine, are introduced.

Experimental evaluation is performed using synthetically generated datasets including various event patterns, investigating the ECMAF's performance and scalability, in terms of execution time and throughput, as well as optimality and accuracy with regards to the produced adaptation strategy. The results show excellent agreement with theoretical the main work assumptions. In particular, the performance of the pattern discovery process mainly relies on the metric's definition, which can be adjusted to reflect the optimal time interval dictated by the pattern discovery process, so as to produce the most effective proactive adaptation results. As far as the performance results are concerned, they show that the ECMAF's execution time depends mostly on the considered monitored metrics, as well as on the deployment's scope (single or multi-Cloud) and SBA's size (i.e. the number of individual services). The accuracy and the performance of the adaptation process (mainly the scaling actions) are mainly based on the image size, the location and the number of the provisioned VMs, but mostly depends on the expertise of the adaptation strategy designer. Finally, the overall ECMAF's evaluation results reveal efficient handling of the detected monitoring events, thus enabling the successful addressing of the individual SLO violations (i.e. reactive adaptation), as well as of the discovered critical event patterns causing aggregate metric's violations (i.e. proactive adaptation).

Keywords: Web Services, Cloud Computing, Service-oriented Architecture, Monitoring, Adaptation, Multi-Cloud, Cross-layer, Modeling

Supervisor: Dimitris Plexousakis
Professor
Computer Science Department
University of Crete

Περίληψη

Τα τελευταία χρόνια, η Υπηρεσιοστρεφής Αρχιτεκτονική έχει αναδειχθεί στον συνηθέστερα χρησιμοποιούμενο τρόπο διάθεσης λογισμικού, με την αξιοποίηση ηλεκτρονικών υπηρεσιών που είναι διάθεσιμες μέσω ειδικών αποθετηρίων στο διαδίκτυο. Προσφέρει πολλές ευκαιρίες αυτοματοποίησης διεργασιών στις επιχειρήσεις, παρέχοντας στους τελικούς χρήστες είτε απλές υπηρεσίες, είτε άλλες πιο σύνθετες, μέσω κατάλληλων διεπαφών. Η επιτυχία των προγραμμάτων που έχουν ως αντικείμενο τις ηλεκτρονικές υπηρεσίες έχει αποδείξει ότι οι υπάρχουσες τεχνολογίες είναι επαρκείς, για να αναπτύξει κάποιος μια κατάλληλη εφαρμογή βασισμένη στην υπηρεσιοστρεφή αρχιτεκτονική. Η εξέλιξη όμως των ηλεκτρονικών υπηρεσιών τα τελευταία χρόνια δείχνει, ότι η τάση είναι όχι τόσο στην απλή ανταλλαγή πληροφορίας μεταξύ της υπηρεσίας και του χρήστη, αλλά στη σύνθεση των υπαρχόντων υπηρεσιών για τη δημιουργία σύνθετων εφαρμογών. Επομένως, οι επιχειρήσεις μπορούν να αναπτύξουν σύνθετες προσαρμόσιμες υπηρεσιοστρεφείς εφαρμογές, εκμεταλλευόμενες παράλληλα παλαιότερες τεχνολογίες που χρησιμοποιούνταν σε τοπικές και απομακρυσμένες εφαρμογές.

Με τον ερχομό του 21ου αιώνα αρκετές νέες τεχνολογίες εισήχθησαν για να συμπληρώσουν και να επεκτείνουν τις δυνατότητες της υπηρεσιοστρεφούς αρχιτεκτονικής, έτσι ώστε να συμβαδίζει με τη νέα τάση του “Διαδικτύου των πραγμάτων” (Internet of Things), το οποίο αναφέρεται στην διασύνδεση των μοναδικών υπολογιστκών μηχανών με την υπάρχουσα υποδομή του διαδικτύου. Το υπολογιστικό νέφος (Cloud computing) είναι μια τέτοια σύγχρονη τεχνολογία που γνώρισε ιδιαίτερη ανάπτυξη την τελευταία δεκαετία και κύριο γνώρισμά του είναι η

παροχή υπηρεσιών σε διάφορα επίπεδα στους τελικούς χρήστες: (i) *Software-as-a-Service (SaaS)* είναι η παροχή εφαρμογών στους χρήστες μέσω του διαδικτύου από τους παρόχους υπολογιστικού νέφους, (ii) *Platform-as-a-Service (PaaS)* αναφέρεται στην παροχή μιας ενδιάμεσης πλατφόρμας στους σχεδιαστές εφαρμογών για να προσφέρουν τις υπηρεσίες τους και (iii) *Infrastructure-as-a-Service (IaaS)* είναι ένα μοντέλο για τις εικονικές μηχανές (virtual machines) που προσφέρουν οι πάροχοι υπολογιστικού νέφους και απαιτούνται από τα άλλα δύο επίπεδα.

Ωστόσο, η δυναμική φυση των ηλεκτρονικών υπηρεσιών και το ευάλωτο περιβάλλον εκτέλεσής τους, επιβάλουν την παρακολούθηση τόσο των λειτουργικών όσο και των ποιοτικών χαρακτηριστικών τους. Αυτή η διαδικασία επιτρέπει επίσης την προσαρμογή των ηλεκτρονικών υπηρεσιών με βάση τις νέες απαιτήσεις που ανακύπτουν κατά τη διάρκεια εκτέλεσής τους. Αυτή η διδακτορική διατριβή εισάγει ένα πλαίσιο εφαρμογής (ECMAF) για την παρακολούθηση και προσαρμογή των εφαρμογών που βασίζονται σε ηλεκτρονικές υπηρεσίες και οι οποίες αναπτύσσονται σε πολλαπλά υπολογιστικά νέφη. Εντούτοις, οι δύο προαναφερθείσες αλληλένδετες αρχιτεκτονικές είναι πολυεπίπεδες και εμπλέκουν ένα σύνολο από συνιστώσες μεγάλου εύρους, από αφηρημένες επιχειρησιακές διεργασίες ως και συγκεκριμένες υπολογιστικές υποδομές. Επομένως, οι διαδικασίες της παρακολούθησης και της προσαρμογής πρέπει να λαμβάνουν υπόψιν όλα αυτά τα επίπεδα, για να μπορούν να χειριστούν αποτελεσματικά τον μεγάλο όγκο των γεγονότων παρακολούθησης και να εξάγουν κατάλληλες στρατηγικές προσαρμογής. Η προτεινόμενη προσέγγιση συνδυάζει τα γεγονότα παρακολούθησης και εξερευνά μοτίβα γεγονότων, τα οποία οδηγούν σε παραβάσεις των ορίων συγκεκριμένων μετρικών, που ορίζονται στα έγγραφα Service-level Agreement (SLAs).

Προκειμένου να ελεγχθεί η εφαρμοσιμότητα και να αναδειχθούν τα πλεονεκτήματα του προτεινόμενου πλαισίου εφαρμογής, στα πλαίσια αυτής της διατριβής υλοποιήθηκε μια εφαρμογή διαχείρισης της κυκλοφορίας στους δρόμους μίας πόλης. Στη συνέχεια αυτή η εφαρμογή αναπτύχθηκε σε ένα περιβάλλον πολλαπλών υπολογιστικών νεφών, τα οποία ικανοποιούν διαφορετικές απαιτήσεις των επιμέρους ηλεκτρονικών υπηρεσιών που απαρτίζουν την εφαρμογή. Τα συστατικά στοιχεία αυτών των υπολογιστικών νεφών έχουν πολλές εξαρτήσεις μεταξύ τους, σε όλα τα επίπεδα της υπηρεσιοστρεφούς αρχιτεκτονικής και της αρχιτεκτονικής υπολογιστικού νέφους. Το προτεινόμενο σύστημα ανακαλύπτει και εκμεταλλεύεται αυτές τις εξαρτήσεις, για να εξάγει από τα γεγονότα παρακολούθησης έγκυρα μοτίβα που επιφέρουν παραβιάσεις συγκεκριμένων μετρικών. Τα τελευταία υφίστανται περαιτέρω επεξεργασία για την εξαγωγή πιο πολύπλοκων μοτίβων, τα οποία αντιστοιχίζονται σε κατάλληλες στρατηγικές προσαρμογής. Για την μοντελοποίηση των εξαρτήσεων έχουμε αναπτύξει ένα ειδικό μετα-μοντέλο, το οποίο αποτυπώνει όλα τα εμπλεκόμενα συστατικά στοιχεία και τις μεταξύ τους σχέσεις. Οι αλληλεξαρτήσεις αυτές εμπλουτίζονται επίσης κατά το χρόνο εκτέλεσης για να αντικατοπτρίζουν τη νέα συμπεριφορά του συστήματος, μετά από αλλαγές που πιθανόν να προήλθαν από μεταβολές στο περιβάλλον εκτέλεσης. Αντίστοιχα μετα-μοντέλα προτείνονται για την επικύρωση της μορφής των γεγονότων παρακολούθησης καθώς και των αντίστοιχων στρατηγικών προσαρμογής.

Για την πειραματική αξιολόγηση του προτεινόμενου πλαισίου εφαρμογής χρησιμοποιούμε τεχνητά σύνολα δεδομένων και ελέγχουμε την κλιμακωσιμότητά του, την απόδοσή του ως προς το χρόνο εκτέλεσης και τη διεκπεραιωτική του ικανότητα, καθώς και την καταλληλότητα της παραγόμενης στρατηγικής προσαρμογής. Τα αποτελέσματα της αξιολόγησης δείχνουν ότι η απόδοση του υποσυστήματος ανακάλυψης μοτίβων γεγονότων παρακολούθησης εξαρτάται κυρίως από τον ορισμό της μετρικής, δηλαδή το χρονικό διάστημα που μεσολαβεί μεταξύ δύο διαδοχικών μετρήσεων. Αυτό μπορεί κατάλληλα να προσαρμοστεί έτσι ώστε να επιτύχουμε τα βέλτιστα αποτελέσματα προληπτικής προσαρμογής της εφαρμογής.

Όσον αφορά την απόδοση του συστήματος τα αποτελέσματα φανερώνουν μια αμέση συσχέτιση με το πλήθος και το είδος των υπό εξέταση μετρικών, το πλήθος των επιμέρους ηλεκτρονικών υπηρεσιών που απαρτίζουν την εφαρμογή καθώς και το είδος της ανάπτυξης της εφαρμογής (σε ένα ή πολλαπλά νέφη). Η ορθότητα και η απόδοση της διαδικασίας προσαρμογής (κυρίως των ενεργειών κλιμάκωσης) επηρεάζεται από το μέγεθος, την τοποθεσία και το πλήθος των παρεχόμενων εικονικών μηχανών, αλλά βασίζεται κυρίως στην τεχνογνωσία και την εμπειρία του

σχεδιαστή στρατηγικών προσαρμογής. Τέλος, τα συνολικα αποτελέσματα του προτεινόμενου πλαισίου εφαρμογής φανερώνουν τον αποδοτικό χειρισμό των γεγονότων παρακολούθησης, δίνοντας έτσι τη δυνατότητα στον πάροχο της εφαρμογής να αντιμετωπίζει επιτυχώς τόσο τις μεμονωμένες παραβιάσεις μετρικών (αντιδραστική προσαρμογή), όσο και των μοτίβων γεγονότων παρακολούθησης που έχουν ανιχνευτεί και προκαλούν παραβιάσεις σύνθετων μετρικών (προληπτική προσαρμογή).

Λέξεις-κλειδιά: Ηλεκτρονικές Υπηρεσίες, Υπολογιστική Νέφους, Υπηρεσιοστρεφής Αρχιτεκτονική, Παρακολούθηση, Προσαρμογή, Μοντελοποίηση

Επόπτης Καθηγητής: Δημήτρης Πλεξουσάκης
Καθηγητής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

To the love of my life

Agapi

Acknowledgements

Pursuing a PhD is indeed a truly long and hard life-changing journey. Fortunately enough, this journey turned out to be a pleasant one thanks to the support of a lot of people, who helped me, to a lesser or greater degree, to accomplish this tough endeavor.

First and foremost, I would like to express my special thanks and appreciation to my supervisor, Professor Dimitris Plexousakis for encouraging my research and for helping me to surpass all research obstacles that came into my way. In addition, his expert guidance and valuable suggestions have greatly contributed to this thesis and laid the foundations of my journey in academia.

I would also like to thank Professor Kostas Magoutis, member of my advisory committee, for his great support and valuable comments and advice during key stages of my PhD thesis, as well as for our effective collaboration in research projects. Furthermore, I wish to thank Professor Christos Nikolaou, serving as an advisory committee member, for giving me the chance to work for various research projects. Moreover, I owe gratitude to the members of my Examination Committee, namely, Schahram Dustdar, Antonio Brogi, Evangelos Markatos and Nikos Parlavantzas, who provided very valuable feedback during my PhD thesis' defense.

In addition, I would like to acknowledge the support of the Institute of Computer Science (ICS-FORTH) and the University of Crete for financially supporting my research during all these five years. Especially, the Information Systems Laboratory (ISL) of ICS-FORTH provided me with all the necessary facilities and an excellent working environment. Particularly, I would like to thank Maria Moutsaki and Dimitis Agelakis for their willingness and continuous support to find solutions to all the administrative and technical problems I encountered during my years at FORTH.

Undoubtedly, I owe a great gratitude to all my friends and colleagues that supported me during my PhD studies. In particular, I would like to especially thank George Baryannis, who was a great "fellow traveller" in this difficult and long journey. Many thanks go to Konstantina Konsolaki, Panagiotis Garefalakis, Damianos Metalidis, Antonis Papaioannou, Antonis Papadogiannakis, Panagiotis Papadakos, Dimitra Zografistou, Roula Avgoustaki and Manos Papadakis for both supporting me in various research topics and also contributing to a fun and relaxing working environment. In addition, I want to thank my friends Tasos Charalambidis and Nikos Valtsis for the great moments we shared, during their stay in Heraklion.

Moreover, I would like to wholeheartedly thank Kyriakos Kritikos, who stood to me as a second supervisor during all the years of my PhD studies. I really appreciate his major support and contribution to this PhD dissertation. I will never forget the fruitful discussions we had in his office, providing, with great patience, valuable feedback and future research directions. Kyriakos thank you.

Finally, I wish to thank my fiancée and love of my life Agapi Perysinaki, for the invaluable love and support that she has unconditionally given to me during the last five years. I really owe her a great gratitude for being with me all these years and has made them the best years of my life. Without a doubt this PhD is worthy devoted to her. In addition, I would like to specially thank her parents, Antigoni and Dimitris, and her sister, Voula, for their support and love.

The greatest gratitude, however, is rightfully owed to my parents, Eleni and Kostas, and my brother, Dimitris, for their endless support and invaluable love and support during all these years. Your prayer for me was what sustained me thus far. I really thank you from the bottom of my heart.

All things are difficult before they are easy.

– Thomas Fuller

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Contents

1.1 SOA and Cloud Computing

The advent of the 21^{st} century has brought *Web services* as a new means of delivering software via a network, such as the internet. This type of software has emerged through the rapid growth of *Service-Oriented Computing* (SOC) [Huhns and Singh 2005], a promising computing paradigm that utilizes services as constructs to support the development of rapid, low-cost composition of distributed applications [Papazoglou 2008]. SOC is essentially a fusion of existing well-established technologies, such as distributed systems, software engineering, Web-based computing and XML technologies, rather than a new technology created from scratch. This technology enables developers to make publicly available their own applications, or discover, invoke and compose existing network-available Web services to offer a new *Service-based Application* (SBA) to the potential users.

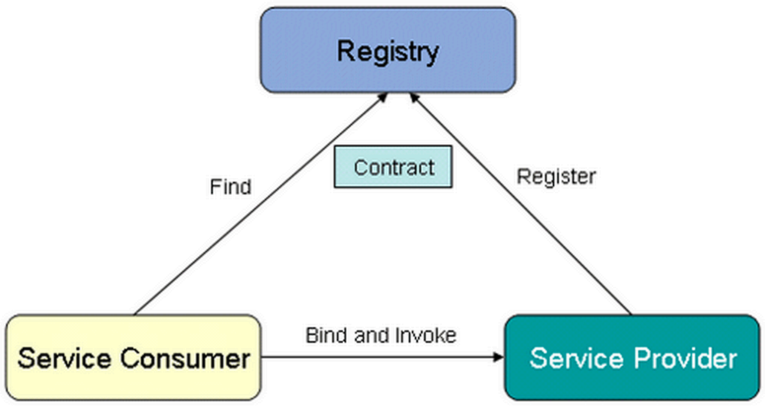


Figure 1.1: The SOA architecture

There three main building blocks comprising the Service-Oriented Architecture (SOA) (Figure 1.1) and they are determined on the basis of three primary roles that can be undertaken by these architectural modules. These are: (i) the service provider, (ii) the service registry; (iii) and the service consumer. Service providers are software agents that provide the service. They are responsible for publishing a description of the service(s) they provide on a service registry. Service consumers are software agents that request the execution of a service. Agents can be simultaneously both service clients and providers. Service consumers must be able to find the description of the services they require and must be able to bind to them. To achieve this functionality, SOA builds on today's Web services standard technology stack (Figure 1.2). The interaction between service providers and consumers is basically loosely coupled, as the service-oriented model does not mandate any predetermined agreements, though most service providers sign *Service-level Agreements* (SLAs) with their customers to guarantee the *Quality of Service* (QoS). Finally, a service registry is a repository that contains Web service related meta information (e.g. Web service descriptions).

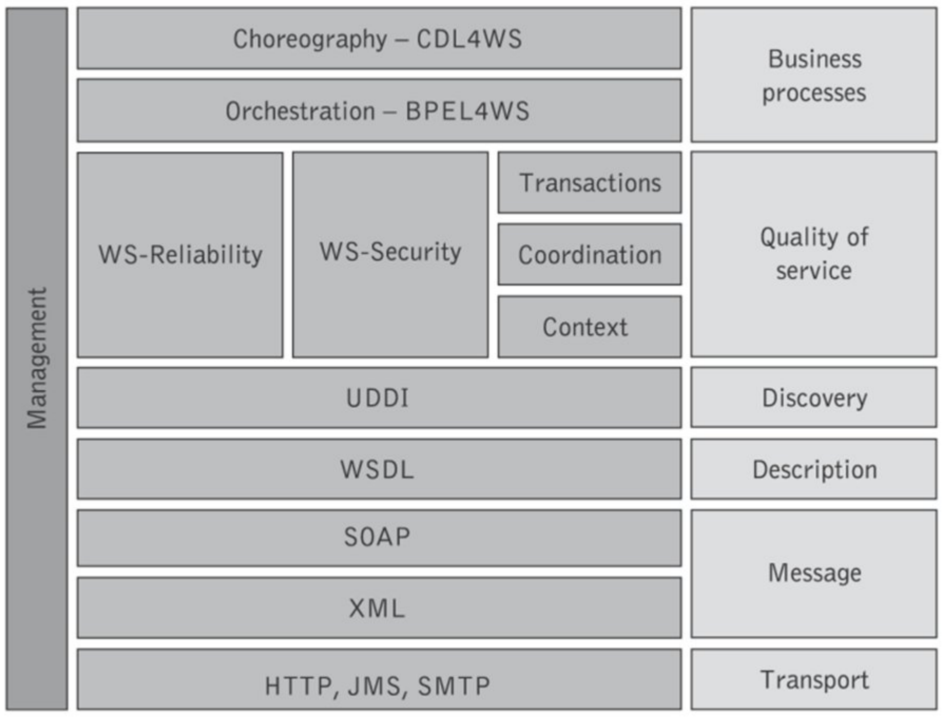


Figure 1.2: The Web services' standard technology stack [Papazoglou 2008]

One major advantage of Web services is that they may either be implemented and distributed on a single machine or a local area network, or even across several wide area networks including a variety of machines, such as computers, mobile devices, etc. In such distributed environments, Cloud computing [Vouk 2008] has emerged in the early 2000s as a new paradigm for delivering "as-a-service" offerings to the end users: (i) *Software-as-a-Service (SaaS)* is the software delivery model adopted in Cloud computing, where Web services are made available to users on demand via the Internet from a Cloud provider, (ii) *Platform-as-a-Service (PaaS)* provides the platform to the application owners to deploy their applications on, and (iii) *Infrastructure-as-a-Service (IaaS)* is a resource provisioning model allowing Cloud providers to outsource computing resources, i.e. *Virtual Machines (VMs)*, needed by SaaS and PaaS. The main notion in Cloud computing is to deliver everything as a service *(XaaS)*, thus one can also come across the following concepts: Security-as-a-service (SECaaS), Communications-as-a-service (CaaS), Network-as-a-service (NaaS), Monitoring-as-a-service (MaaS) and Data-as-a-Service (DaaS).

These two computing paradigms are closely interrelated, as Web services are the main SaaS type deployed on Cloud infrastructures, while the virtualized Cloud resources provide an efficient application delivery platform for the SBA developers. As the author in [Fernandez 2012] point out, Cloud computing is a new embodiment of SOC, where users access a variety of services to implement their functional needs. Instead of providing users with application-oriented services, now users can access services on any architectural level, using the Cloud as the conduit with the primary *Cloud SBA*. In practice, Cloud computing provides the computing of services and SOC provides the services of computing [Wei and Blake 2010].

This reciprocal relationship between SOC and Cloud computing paves the way to many research directions within the lifecycle of Cloud SBAs, such as infrastructure description, requirements specification, service description, discovery, matchmaking, composition, Cloud SBA deployment, execution, monitoring, adaptation and evolution (see Figure 1.3). This dissertation focuses on research issues related to monitoring, adaptation and partially evolution of SBAs in Cloud environments.

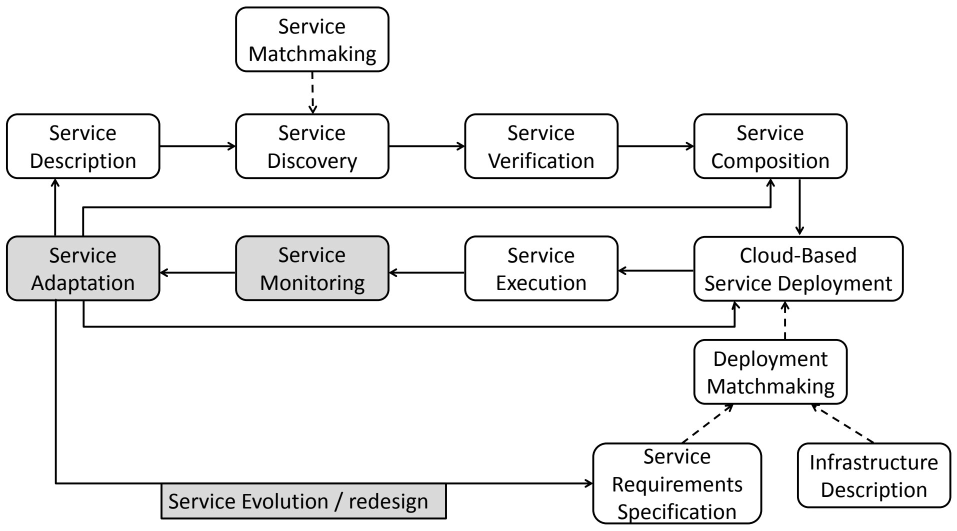


Figure 1.3: The lifecycle of a Cloud SBA

1.1.1 SBA Layers

A SBA [Kazhamiakin et al. 2009a] is composed by a number of possibly independent services, available in a network, which perform the desired functionalities of the architecture. Such services could be provided by third parties, not necessarily by the owner of the SBA. Note that a SBA shows a profound difference with respect to a component-based application: while the owner of the component-based application also owns and controls its components, the owner of a SBA does not own, in general, the component services, nor it can control their execution.

A SBA can be represented by its three functional layers (Figure 1.4), as they have been adopted by many EU projects, such as S-Cube[1] and SOA4All[2]: (i) the business process management (**BPM**) layer, (ii) the service composition and coordination (**SCC**) layer; and (iii) service infrastructure (**SI**). BPM is the highest level functional layer where the entire business process is defined along with the application activities, constraints and requirements without going into design details. Henceforth, we consider the entire business process as a workflow and the business activities as its constituents. Moreover, at the BPM layer, Service Networks (SNs) [Danylevych et al. 2011] are defined in order to enable companies build networks to serve their joint customers in a dynamic manner, focusing on optimizing their financial benefits at the individual and network level and the company's business collaborations. SCC is the layer between BPM and SI layers, where the basic workflow constructed at BPM is refined by the composition of suitable services, that is capable of realizing the corresponding business activities. This layer organizes and manages the control and data flows among services in the composition. Finally, the SI layer provides the underlying run-time environment for the composed SBA. This environment manages a large set of computing resources, such as storing and processing capacity. In addition, the collection of all available services is kept in a service registry provided by this layer. Consequently, the identified services for the composition are discovered and realized at SI level.

[1]http://www.s-cube-network.eu/

[2]http://www.soa4all.eu/

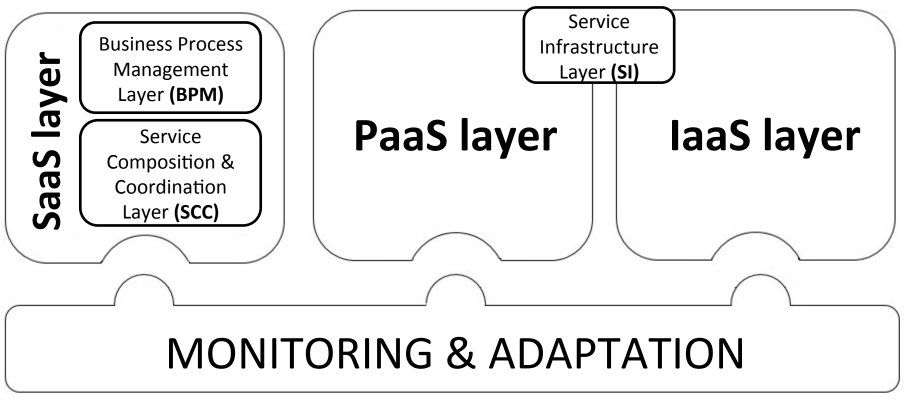

SaaS layer
Business Process Management Layer (BPM)
Service Composition & Coordination Layer (SCC)
Service Infrastructure Layer (SI)
PaaS layer
IaaS layer
MONITORING & ADAPTATION

Figure 1.4: The layers of a Cloud SBA

1.1.2 Cloud Deployments

This section analyzes the different Cloud deployments available to the Cloud providers, pinpointing their main benefits and drawbacks. Cloud computing technology provides large pools of resources that can be connected through private or public networks, as well as dynamically scalable infrastructure for cloud based applications, data and file storage. Businesses during the decision-making process of their ideal Cloud deployment may choose to deploy applications on *Public*, *Private* or *Hybrid* Clouds. Figure 1.5 depicts these deployments and their interrelations.

First, as stated in [Mell and Grance 2011], *a public Cloud is provisioned over a network that is open for public use. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them, but in any case it resides on the premises of the cloud provider.* This model provides no visibility or fully control over the underlying infrastructure to the customers, who share the same infrastructure pool with limited configuration, security protections and availability variances. The main advantage of a Public Cloud is its cost-effectiveness, as infrastructure costs are spread across all users, allowing each individual client to operate on a low-cost, pay-as-you-go model. Another advantage is its on-demand

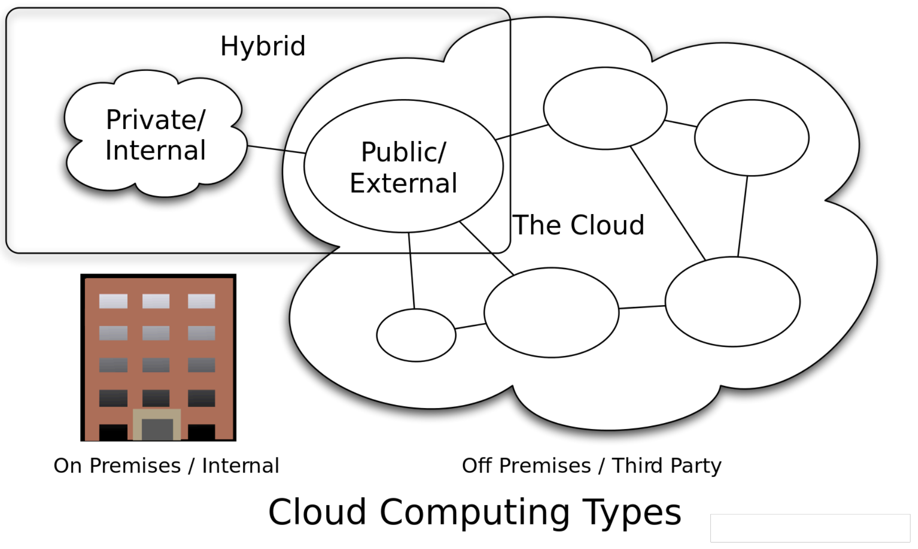


Figure 1.5: The Cloud deployment types

and higher scalability than an in-house enterprise cloud. In a nutshell, these clouds offer the greatest level of efficiency in shared resources; however, they are more vulnerable and less secure than private clouds. They are the ideal solution [Josh 2012] when: (i) the SBA is used by a great amount of customers, (ii) there is a need for testing and developing application code managed by many developers within collaborative projects; and (iii) there are high-scalability demands.

Second, *a Private Cloud is provisioned for exclusive use by a single organization comprising multiple consumers and may be owned, managed and operated by the organization, a third party, or some combination of them and it may exist on or off premises* [Mell and Grance 2011]. Private clouds allow businesses to host applications in the Cloud, while addressing concerns regarding data security and control, which is often lacking in a public cloud environment. Undertaking a private cloud project requires a significant effort to virtualize the business environment and it will require the organization to re-evaluate decisions about existing resources. Compared to the public Cloud, they are undoubtedly more expensive, but also more secure and fully-controlled and manageable. It is the ideal solution when: (i) Cloud efficiency and control is a high-priority, (ii) there is a need for consistency among services, (iii) the provided services are private and used only within the enterprise premises; and (iv) there is redundant server capacity.

Third, *a Hybrid Cloud is a composition of two or more distinct Cloud infrastructures (private or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability* [Mell and Grance 2011]. Its main advantage is that it enables increasing the flexibility of computing by leveraging third party cloud providers. Moreoever, optimizing a private Cloud with public cloud resources can prevent unexpected failures and resource overloading. This cloud deployment requires both on-premise resources and off-site server based Cloud infrastructure. Its main drawback is that you have to keep track of multiple Cloud security platforms and ensure successful communication between private and Cloud platforms. It is the best solution when (i) the provided SaaS applications are tailored for different vertical markets, (ii) the provided application requires both high security and scalability; and (iii) one can provide public Cloud to her/his customers while using a private cloud for internal IT.

1.2 Motivation

In the context of the Cloud SBA's lifecycle depicted in Figure 1.3, *monitoring* refers to the process of collecting and reporting relevant information about the execution and evolution of a SBA. This information is exploited by the *adaptation* process, that modifies SBAs in order to satisfy new requirements and to fit new situations dictated by the environment on the basis of adaptation strategies designed by the system integrator. *Evolution* is a long-term history of continuous modification of a SBA (i.e. redesign) after its deployment in order to correct faults, to improve performance or other attributes, or to adapt it to a modified environment [Benbernou et al. 2008].

Once services and business processes become operational, their progress needs to be managed and monitored to gain a clear view of how services perform within their operational environment, take management decisions and perform control actions to modify and adjust their behavior. The dynamic and ever-changing nature of the business and physical environment requires Web services to be highly reactive and adaptive to the changes and variations they are subject to. They should be equipped with mechanisms to ensure that they are able to adapt to changing requirements. Web service monitoring and adaptation are two well-connected processes that result in the correction and customization of faulty services, so as to coincide with the new requirements.

Furthermore, as Cloud computing technologies have been maturing, enterprises are increasingly adopting Cloud platforms to offer their SBAs. In such distributed virtualized environments, the SBA developers, in an effort to optimize their applications deployment cost and performance, may also deploy application parts redundantly on different VMs that may be offered by different Cloud providers. In [Baryannis et al. 2013b] we have pinpointed the main benefits of SBA multi-Cloud deployments: (i) different resource requirements are better satisfied by different Cloud providers offering special features (e.g., SSD drives, dynamically configurable VMs), (ii) the deployment cost is optimized by exploiting the Cloud providers variant cost policies; and (iii) cross-talk components are deployed on the same geographical zone or even VM to minimize communication overhead.

In such heterogeneous environments, where multiple Cloud providers are involved, an important concern for any Cloud-based application provider is to maintain its desired level of service along its entire life cycle, as well as to have a clear view of the system's state, the system component interrelationships and the performance of the underlying infrastructure [Papadogiannakis et al. 2012]. In addition, during application execution, various events are produced by several layers (Cloud and SOA specific), leading or indicating Service level Objective (SLO) violations. Just as in any distributed application hosting environment, Clouds must support extensive monitoring mechanisms to aid in controlling application performance and functionality and to adapt to variations, mainly at the IaaS layer. An effective monitoring mechanism must cover the entire range of Cloud SBA layers (Figure 1.4). Leading Cloud providers (e.g. Amazon AWS, Microsoft Azure, HP Cloud, Rackspace, etc.) are just starting to roll out solutions in both areas, while most of the industry is still lagging behind. Looking forward, applications designed for multi-Cloud environments will be facing challenges stemming from the lack of uniform (cross-platform) support for monitoring and adaptation solutions. Assuming that these challenges are eventually met, a number of other problems still need attention: (i) cross-layer (IaaS, PaaS and SaaS) monitoring and alignment of the monitoring events; (ii) cross-layer coordination of adaptation actions; and (iii) proactive as well as reactive adaptation policies.

Definition 1 *: **Monitoring event** is a completion event, i.e. the corresponding metric's assessment result.*

Concerning cross-layer monitoring and adaptation [Treiber 2009], current solutions serving SBA monitoring are mostly constrained to one layer or to very specific aspects, such as process metrics as part of business activity monitoring (i.e., KPIs), or QoS metrics as part of SLA monitoring (i.e., SLA metrics) and do not integrate and correlate information from all layers. Respectively, SBA adaptation approaches mostly rely on layer-specific adaptation actions, ignoring the implications that may be posed by their application on other layers. As such, as composite services implement business processes from the BPM layer and at the same time are based on QoS properties at the SCC layer and IT infrastructure properties at the SI layer, it is imperative that monitoring and adaptation of SBAs take into account all the functional layers.

Considering the close relationship between Cloud and SBAs layers, it becomes important to perform and correlate monitoring across all layers. While it is hard to overestimate the value of effective monitoring (strong control over infrastructure, support for elasticity policies and QoS, etc.), most current monitoring approaches are fragmented [Zeginis 2009] (confined within a Cloud provider or specific service layers) and are not applicable or aligned across layers. Specifically, they do not relate events reported by different layers, triggering disassociated (and often conflicting) adaptation actions for each such event. Multi-Cloud deployments of SBAs further complicate this picture due to the lack of cross-platform support for uniform monitoring solutions.

Although current approaches cover a wide spectrum of monitoring and adaptation, none of them can efficiently cover all service and Cloud layers. They are usually considered in isolation from each other and focus on a local solution for a specific monitoring or adaptation requirement without taking into account the effects to the other SBA layers. While they seem to be quite effective on a specific layer when considered in isolation, they can cause problems and incompatibilities to the other layers when adaptation actions take place. Different artifacts (i.e. components) at one layer may refer to the same artifacts of another layer, while such relations are ignored by the isolated monitoring and adaptation solutions. As a consequence, wrong problems are detected, incorrect decisions are made and the modifications at one level may damage the functionality of another layer.

Furthermore, the plethora of monitoring event types produced in complex multi-Cloud systems requires rich and extensible meta-models that can describe every single event detected by the monitoring mechanisms, containing not only non-functional (QoS) but also functional aspects of the service. These events are usually causally interrelated forming event patternsC that lead to specific SLO violations. In principle, these interrelationships stem from the SBA component dependencies [Magoutis et al. 2008], thus giving rise to the need for a component meta-model able to capture these dependencies for a multi-cloud SBA, as well as define the adaptation capabilities of the involved components. Finally, the adaptation systems emerging in such multi-Cloud environments should be equipped with special adaptation models, describing the adaptation strategies, as well as the lower level actions that can be performed by the existing adaptation enactment mechanisms.

As a result, the main research questions that arise from the above analysis and are answered in this dissertation are the following:

- How can we efficiently perform cross-layer monitoring and adaptation of multi-cloud SBAs?
- How can we gradually and flexibly discover monitoring event patterns that are interrelated with specific SLO violations?
- How can we define the most appropriate and cost-effective adaptation strategy for a monitoring event pattern?
- How can we formally describe and model the monitoring events, the component dependencies and the adaptation actions for a multi-cloud SBA?
- How can we discover a particular root problem through the exploitation of component/service dependencies?

1.3 Monitoring and Adaptation Lifecycle

This section analyzes the processes constituting the SBA lifecycle presented in Figure 1.3. The main focus of this thesis is on Web service monitoring and adaptation (MA), thus more details are provided for these two processes.

The need for two cycles [Benbernou et al. 2008] arises from the dynamic nature of Web services and the vulnerable execution environment in which they perform. These cycles co-exist and do not conflict with each other. Particularly, in the *requirements specification and design* phase, the MA requirements are identified, while during *SBA construction*, the corresponding MA mechanisms are being realized, together with the construction of the SBA. During the *operation, monitoring and management* phase, run-time monitoring is being executed, based on the monitored properties defined in the contracted SLA document. Thereafter, two different paths are available: *evolution* and *adaptation*. The former path dictates restart of the right-side of the cycle comprising the requirements engineering and design phases, while the latter one involves the *identification of the adaptation needs*. Each adaptation need maps to a specific adaptation strategy (*adaptation *strategy identification* phase), which in turn leads to the enactment of lower level adaptation actions (*adaptation actions enactment* phase).

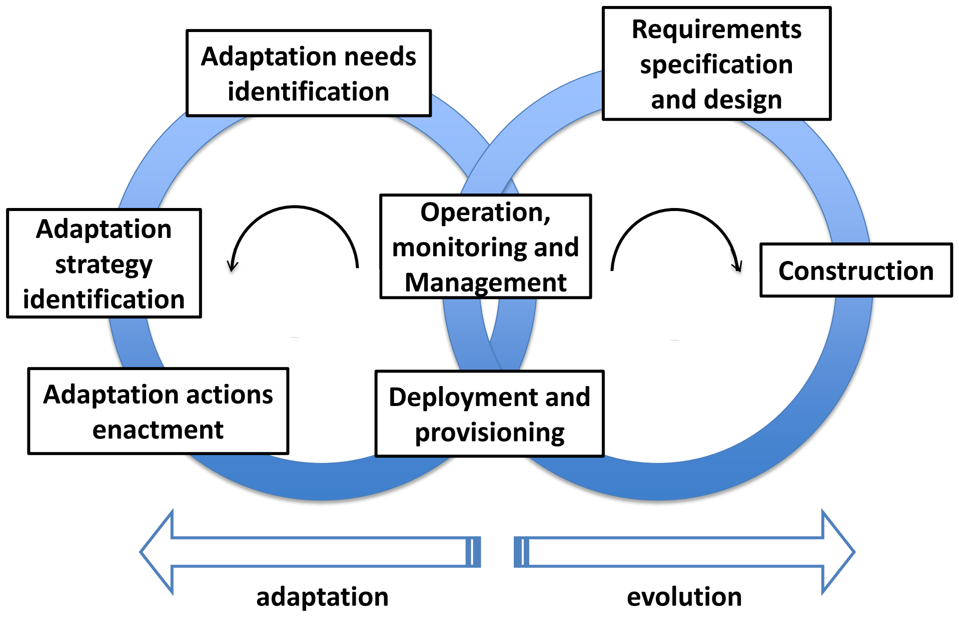

Adaptation needs identification
Requirements specification and design
Adaptation strategy identification
Operation, monitoring and Management
Construction
Adaptation actions enactment
Deployment and provisioning
adaptation
evolution

Figure 1.6: Web service adaptation and evolution

1.4 Requirements

This section unfolds a set of requirements that must be satisfied, so that cross-layer monitoring and adaptation are successful, accurate and complete in multi-cloud environments. The identified requirements, will be used in the following chapters in order to compare the related work against these requirements and to design and implement our contributions according to them.

Based on the application requirements and key quality properties, it is necessary to define the requirements and objectives that should be satisfied when certain discrepancy is detected with respect to the expected SBA state functionality or environment. More precisely, the monitoring requirements specify what should be observed and when the discrepancy becomes critical for the SBA. The adaptation requirements describe the desired situation, state, or functionality, in which the SBA should be brought. Typically, the monitoring and adaptation requirements correspond to various SBA quality characteristics that range from dependability, to functional, behavioral correctness and usability. In many cases monitoring requirements derive directly from the adaptation requirements: monitoring is often performed in the aim of identifying the need for adaptation and triggering it.

After reviewing the related work (see Chapter 3) we identified and came up with a number of requirements that must be satisfied in order for monitoring and adaptation to be successful, accurate and complete. These requirements, that will be referred throughout this thesis are used to pinpoint the strengths and benefits of our approach. They also form the base of the related work comparison. The rest of this section provides a brief overview of these requirements:

The primary requirement that must be satisfied is the *cross-layer monitoring and adaptation capability*, dictating that the approach should take into account both the SOA and Cloud layers comprising a Cloud SBA, as they are identified in Section 1.2. Other requirements, acquiring different meaning towards monitoring and adaptation, include *dynamicity*, *intrusiveness*, *timeliness*, the *kind of properties* they address and their *scope*. The dynamicity of a monitoring framework concerns the ability of the framework to change monitoring properties during the execution of the process, whereas the dynamicity of an adaptation framework allows additions and deletions of adaptation rules. Moreover, approaches which perform monitoring by weaving code that implements the required checks inside the code of the system that is being monitored, are concerned as intrusive approaches. Thus, the monitoring mechanism should incur low overhead on the nodes it operates on and should also require low bandwidth to transmit the gathered information. Regarding adaptation approaches, a framework is intrusive, when the applied adaptation actions change the process. On the one hand, the timeliness of the monitoring system presents the ability of the framework to signal violations of the monitoring properties as soon as they occur and not after the termination of the instance. On the other hand, timeliness of an adaptation framework refers to the proactive or the reactive execution of the adaptation actions. Furthermore, the kind and the scope of the monitored information is provided. The former one refers to the functional and non functional properties of a SBA, while the latter one refers to the instance or class application of the approach.

Additional requirements are posed when monitoring and adaptation are applied on Cloud systems [Villari et al. 2012]: *multi-tenancy* refers to the ability of the monitoring and adaptation system to support multiple tenants, as Clouds are inherently designed to be used by multiple users. These systems must ensure that monitoring information from particular tenants cannot be accessed by other tenants. Thus, every tenant must have the illusion that she/he is the exclusive user of the system. *Dynamicity and extensibility* in Cloud environments dictates that MA systems be able to adapt a changing set of users and resources. Independently of the already installed monitoring probes and adaptation mechanisms, the MA system should allow for the federation of new ones without interfering with the existing components. In addition, as there is a demand for a variety of different VMs coming from different Cloud providers, *scalability* should be a top requirement for MA systems, specifically in terms of the number of physical and virtual resources and the number of tenants. Scalability in Cloud computing is not the same with elasticity, which spans across three main layers: (i) the resource elasticity, (ii) the quality elasticity; and (iii) the costs and benefit elasticity [Dustdar 2014]. *Simplicity* also requires that MA systems be simple and user-friendly, so that as potential Cloud users can easily install, maintain and provide them to their customers.

1.5 Thesis Contribution and Impact

The key points and contributions of this dissertation that tackle the above research questions are the following:

- It introduces a cross-layer monitoring and adaptation framework for multi-Cloud SBAs, along with implementation details and evaluation results.

- It presents a novel pattern discovery algorithm, that is utilized by the proposed framework to identify event patterns related to violations of specific SLOs.
- It presents a technique for efficiently mapping the discovered event patterns to suitable adaptation strategies, focusing on their appropriateness and cost-effectiveness.
- It presents the following monitoring and adaptation related meta-models:
  - An event meta-model that describes the monitoring events produced during the execution of SBAs in multi-Cloud environments and it is exploited to verify the monitoring events and collect the relevant data for the pattern discovery and the adaptation processes.
  - A component meta-model that defines the dependencies of the components comprising a multi-Cloud SBA. Instances of this meta-model describe current SBA component dependencies and capture a system snapshot at a particular time point. They are also exploited by the event pattern discovery algorithm to identify valid event patterns.
  - An adaptation actions meta-model defining the adaptation actions that can be applied at each layer for a particular Cloud-based SBA. This meta-model can be exploited by any adaptation manager during the mapping from simple monitoring events to suitable adaptation actions.
- Finally, an extensive experimental evaluation of the framework is performed, in order to investigate performance scalability in terms of execution time and memory consumption, as well as accuracy and optimality with regard to the captured monitoring events, the discovered event patterns and the extracted adaptation actions. Evaluation relies on datasets produced during the execution of a traffic management application, used throughout the dissertation as a running example. Results show that good performance

and accuracy is guaranteed, even for stressing intervals producing a great amount of monitoring events.

The proposed monitoring and adaptation framework can have a substantial impact to SOA stakeholders, especially to those who resort in multi-Cloud deployments of their SBAs. This framework enables them to have a clear view of how their applications perform in such distributed virtualized environments, where they do not have direct access to the underlying resources (i.e. hosting machines). Moreover, the incorporated pattern discovery mechanism assists them in performing a root cause analysis of the monitoring events stream and identifying the main source of malfunctions in this multi-tier environment.

The time synchronization techniques and the time-series database (TSDB) utilized by the monitoring engine enable application developers to track the monitoring execution history, maintaining the actual order of the monitoring events, and thus reliably discover their causal relationships. Concerning SBA adaptation managers, they can take advantage of the proposed framework, as it provides semi-automatic adaptation capabilities that can be exploited to define the desired mapping from simple SLO violations to suitable adaptation actions provided by the available adaptation mechanisms. Additionally, the incorporated event pattern discoverer enables injecting proactiveness in a SBA adaptation engine via mapping event patterns to respective adaptation strategies (involving other lower level adaptation actions) through an automatic mapping mechanism, thus lowering the adaptation cost and at the same time making SBAs more attractive to potential customers.

Furthermore, the proposed meta-models describing the monitoring events, the system components and the adaptation actions can be of great assistance to SBA stakeholders, especially SBA designers and managers. In particular, taking advantage of these models, a SBA designer can semi-automatically define the whole spectrum of multi-Cloud SBA components, their dependencies, as well as their adaptation capabilities, thus reducing the design effort. The model component instances can be exploited by any SBA manager, along with the adaptation action meta-model to validate the extracted monitoring event patterns, as well as to efficiently map SLO violations to suitable adaptation actions. Finally, the event meta-model enables the SBA monitor manager to keep the monitoring events, produced by the various monitoring mechanisms installed in the considered multi-Cloud environment, aligned and well-formed.

The novelty of our approach, in comparison to the related work presented in Chapter 3, lies in the following contributions. First, we propose an adaptation-aware component meta-model that exploits existing layer-specific standard models to describe the components of a Multi-Cloud application along with their interrelationships. The component classes of the proposed UML model are carefully designed to assist the mapping of event patterns to specific adaptation actions, by meticulously describing the exact dependencies of each component. Second, we propose a novel logic-based algorithm for discovering event patterns leading to critical events within a monitoring events stream of a multi-cloud application. This algorithm can be used by any Multi-Cloud application developer to discover detrimental patterns causing specific violations and thus enables proactiveness in adapting applications. Third, it introduces, a novel monitoring and adaptation framework for Cloud SBAs. that can efficiently handle the huge amount of monitoring events detected at run-time. The processing of the monitoring event stream results in the triggering of suitable adaptation rules addressing the critical events (reactive adaptation) and the critical event patterns (proactive adaptation).

1.6 Dissertation Outline

The rest of this document is structured as follows. Chapter 2 analyzes a running example that will be used throughout this dissertation, while Chapter 3 provides the required background information on Web service monitoring, adaptation focusing on their cross-layer and multi-cloud aspects, event pattern discovery and Cloud modeling, followed by an overview of the corresponding solutions that have been proposed in literature. A comparative analysis highlighs the main novelties of our work and how it advances state of the art. This chapter concludes with an analysis of the required technical background.

Chapter 4 provides an overview of the dissertation's main contribution, describing the proposed monitoring and adaptation framework, named ECMAF (i.e. **E**vent-based **C**ross-layer **M**onitoring and **A**daptation **F**ramework) [Zeginis et al. 2011], that has also been extended to support multi-cloud SBAs [Baryannis et al. 2013a, Zeginis et al. 2012b]. Chapter 5 presents the proposed meta-models for describing the monitoring events, the components comprising a Cloud SBA and the adaptation actions. Chapter 6 introduces the proposed event pattern discovery algorithm [Zeginis et al. 2014a;b] for identifying event patterns related to specific SLO violations, while Chapter 7 examines how the proposed framework supports both reactive and proactive adaptation. Chapter 8 provides implementation details for the traffic management application (i.e. the running example), the proposed ECMAF framework, as well as for the realization of the proposed meta-models. A thorough evaluation of ECMAF is conducted and the individual components it comprises are presented in Chapter 9. Finally, Chapter 10 summarizes the main contributions of this thesis and points out future research directions.

The research activity related to this dissertation has so far produced 3 conference and 2 workshop papers, 1 book chapter, 1 PhD symposium paper, along with 2 technical reports and 3 presentations at the PhD and poster sessions of the SummerSoC summer school series[3], which are briefly described below, while a journal paper is under submission.

- An initial version of the background analysis and the state of the art along with the main related research challenges presented in Chapter 3 appear in [Zeginis and Plexousakis 2010b] and [Zeginis and Plexousakis 2010a].
- An initial version of the ECMAF framework (Chapter 4), explaining its main

[3]http://www.summersoc.eu

capabilities and functionality and the event meta-model is presented in [Zeginis et al. 2012a].

- The proactive adaptation capabilities offered by the proposed framework presented in Chapter 7 appear in [Zeginis et al. 2012b].
- The Cloud extension of ECMAF and its collocation in a multi-Cloud environment have been published in [Zeginis et al. 2013a] and [Baryannis et al. 2013a].
- The proposed event pattern discovery algorithm in Chapter 6, the proposed component meta-model (Chapter 4) and part of the implementation and evaluation results presented in Chapters 8 and 9 appear in [Zeginis et al. 2014a] and in [Zeginis et al. 2014b].
- The whole multi-Cloud version of ECMAF with implementation details and extensive evaluation results, as well as the adaptation action meta-model is under submission in *IEEE Transactions in Service Computing* journal.

Chapter 2

Running Example

Contents

This chapter details a traffic management (TM) SBA, referred to throughout the rest of this dissertation to exemplify and concretize the functionality and usability of the proposed framework and its involved components. It is a real application, realized with the most prominent SOA and business processing tools and techniques (see Chapter 8). The rest of this chapter analyzes the TM's business process activities and how they are realized by specific composite Web services, as the multi-Cloud deployment of the application.

2.1 Traffic Management Running Example

This traffic management application simulates a TM system for the city of Heraklion and and handle both normal and critical situations. The stakeholders in this case study involve:

- The **Traffic Manager**, who controls the whole application, adjusting the environmental and traffic metrics' thresholds.
- The **Rescue Forces**, i.e. the Traffic Police and the Fire Brigade.
- The **Medical Forces** that are responsible for carrying out manual activities to realize the adaptation actions dictated by the Assessment task.

Figure 2.1 depicts the workflow of the application that consists of three Web services: the Monitoring Service (Task T_M), the Assessment Service (Task T_A) and the Device Configuration Service (Task T_D).

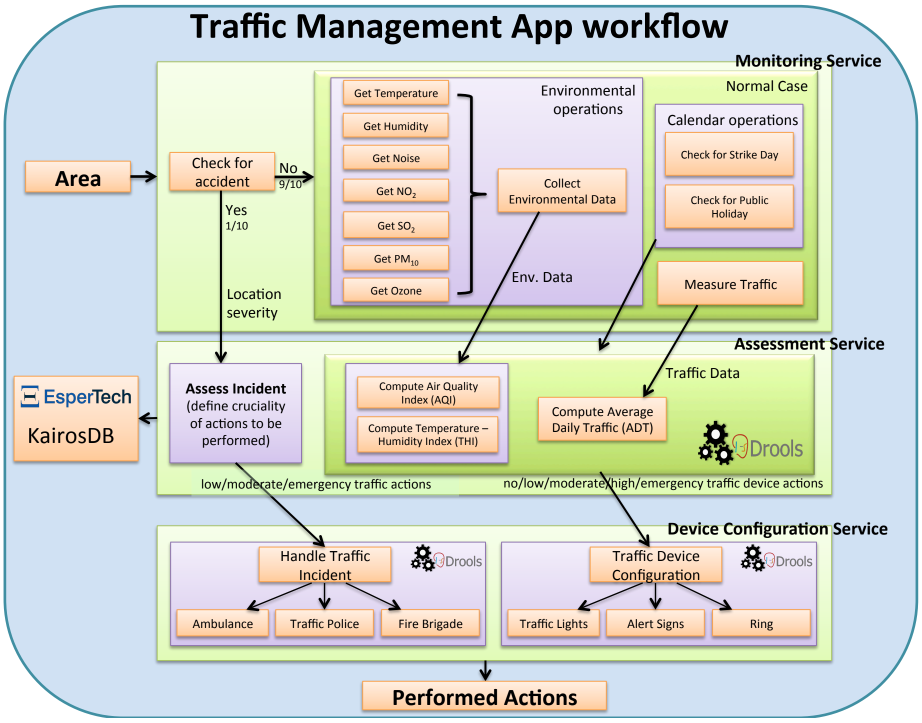


Figure 2.1: The Traffic management application workflow

2.1.1 Monitoring Task

This task is responsible for collecting traffic and environmental data from a specific city area. This application is specifically implemented for the city of Heraklion, which is divided into three main areas; (i) 'area1' is the city center, (ii) 'area2' is the remaining area of Heraklion, except the city center; and (iii) 'area3' comprises the Heraklion suburbs. We assume that pollution is decreasing while moving away form the city center; thus the sensors provide appropriate values for the considered area.

Initially, the interested party passes as input to this service the area for which she/he wants to get monitoring data. Then, the service checks with a probability of 10% if a car accident has happened in this area. On the one hand, in such as critical situation, the concrete location as well as the severity of the accident are passed to the assessment service for further processing. On the other hand, when there is no car accident, the traffic management application regulates the traffic at the specific area taking into consideration the traffic flow density (i.e. the number of cars passing from a specific point in a 24-hour base), the pollution-related environmental variables (i.e. temperature, humidity, noise, ozone level, nitrogen dioxide level (NO_2), Sulfur dioxide level (SO_2)), as well as public events, such as public holidays and strike days. All these data is collected by special sensors and is passed to the corresponding functions of the assessment service. Especially, regarding the environmental data, an environmental object containing this information is constructed, as a prerequisite for computing pollution indices.

2.1.2 Assessment Task

The assessment task is the most computational demanding one, as it collects this big amount of data from the monitoring task and is responsible for assessing its unique set of data and deciding on the actions that should be taken to regulate the traffic appropriately. This task comprises two main functions: (i) the incident assessment function that takes into consideration the severity of the action and decides on the criticality (i.e. low, moderate or emergency) of the traffic actions, basically manual activities that should be performed by the public authorities; and (ii) the normal case assessment functions, which take as input the traffic, environmental and calendar events and decide on the actions, if any, that should be performed to decongest the traffic at the specific area and decrease the pollution levels. In the latter case, the assessment task decides on the criticality (i.e. no, low, moderate, high or emergency) traffic device actions that should be performed by the Traffic Manager in order to regulate the traffic. In high and emergency levels the Traffic Manager decides on additional traffic actions for the public authorities.

In particular, the assessment service comprises three functions for assessing the normal case scenario, based on the environmental, traffic and calendar input events:

- The *Compute Air Quality Index (AQI) function* computes the regional AQI (Equation 2.1), which is an indicator of the aggregate pollution level of four pollutants, namely, SO_2, NO_2, PM_{10} and O_3. C_i means the daily average concentration while R_i represents the daily average concentration limit of the corresponding pollutants[1].

$$AQI = \sum_{i=1}^{4} \frac{C_i}{R_i} \tag{2.1}$$

- The *Compute Temperature-Humidity Index (THI) function*, computes the combination of temperature and humidity as a measure of the degree of discomfort experienced by an individual in warm weather; it is originally called the discomfort index (Equation 2.2). The formula used to compute THI is based on the following formula [Yousif 2013], where *T* is the temperature and *RH* is relative humidity measured by the corresponding sensors.

$$THI = (1.8 * T + 32) - ((0.55 - 0.0055 * RH) * (1.8 * T - 26)) \tag{2.2}$$

[1]http://www-app.gdepb.gov.cn/raqi3/capi_ENG_detail.html

- The *Compute Average Daily Traffic function* computes a traffic-related index that indicates the level of traffic congestion in this area. In particular, this index takes into consideration the number of cars passing a specific point of the area's main roads and computes an average car counter for a specific time interval, aiding the selection of traffic regulation actions. In practice, average daily traffic (ADT), is the average number of vehicles two-way passing a specific point in a 24-hour period, normally measured throughout a year[2].

2.1.3 Device Configuration Task

The Device Configuration Task performs the adaptation actions dictated by the assessment task. This task also employs a Rule Engine to decide on the specific actions to be performed, based on the criticality level dictated by the assessment task. In particular, this task consists of two main set of actions:

- The first set of actions handles the critical situation of a traffic accident occurrence. These manual actions are mainly performed by the traffic police to regulate the traffic at the incident's location and the fire brigade or medical forces for more serious accidents. The alert signs of the affected area are updated when such actions are performed in order to inform the drivers.

- The second set of actions handles normal case situations, i.e., when environmental, traffic levels and/or the occurrence of public/strike days may require traffic devices configuration to decongest the area's traffic. However, they may also be performed as part of a car accident's handling, if it is required by the Traffic Manager. Particularly, these actions involve the configuration of the traffic lights (e.g. blinking orange to alert the drivers), enabling/disabling the area's ring to prevent a number of cars (based on

[2]http://www.rms.nsw.gov.au/publicationsstatisticsforms/trafficvolumes/index.html

their odd/even sign number) from entering the affected area, and the update of the alert signs, when something strange happens (i.e., either accident or environmental/traffic/calendar-related events).

2.1.4 Application Requirements

Tasks T$_M$, T$_A$ and T$_D$ [Baryannis et al. 2013b] have their own resource requirements. Task T$_A$ has high computation, availability and throughput requirements, whereas tasks T$_M$ and T$_D$ do not. T$_M$ and T$_A$ require storage capacity while T$_D$ does not. T$_A$ has moderate storage throughput capabilities, given the fact that, while a high amount of information is exchanged, this happens rather infrequently. Finally, T$_M$ and T$_D$ should be deployed geographically close to the sensor infrastructure (the urban areas) while T$_A$ can be deployed anywhere. Based on these requirements, a multi-Cloud deployment (Figure 2.2) can be created that places T$_M$ and T$_D$ on a private/municipal Cloud (different instances in different cities) close to the sensor infrastructure. The chosen VMs can be of low computational power; T$_M$ however should be coupled with a storage service. T$_A$ can be placed in any Cloud that can provide high-computation VMs with storage capacity at the best reliability/price ratio. The above requirements are summarized in Table 2.1.

Table 2.1: Requirements of the traffic management SBA

Involved Tasks	**Requirement**
T_A	High computational power
T_A	High availability
T_A	High throughput
T_A	Moderate storage throughput
T_M and T_A	High storage capacity
T_M and T_D	Geographically close

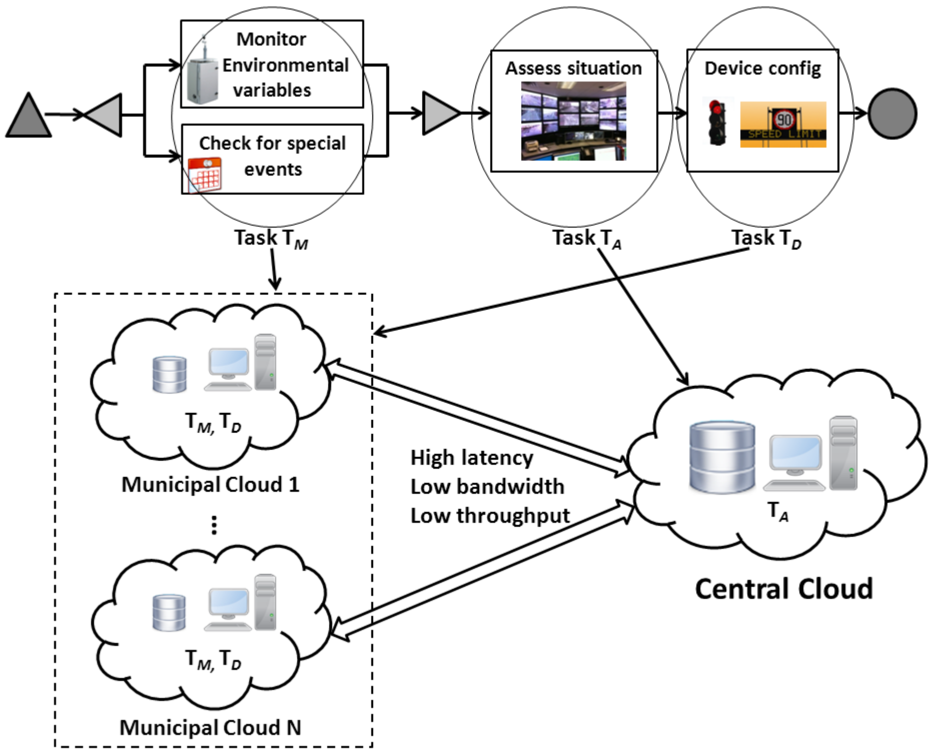


Figure 2.2: Multi-Cloud deployment of the Traffic Management Application

2.2 Application Components

In this subsection we describe a traffic management scenario, that will be referred to throughout the dissertation, in order to exemplify the proposed framework's functionality. The application components involved in this scenario in each layer are the following:

BPM layer: The traffic management Business Process comprises three main activities (Monitoring, Assessment, Device Configuration). These three activities are connected sequentially through the corresponding data and control flows. In order to trigger the execution of the Business Process, the T_M task requires as input a Heraklion city's area. The output of this task maps to the input of the T_A activity and the output of the T_A task maps to the input of the T_D task. Finally, the T_D task outputs the actions that have been performed after a single execution of the business process. Sample BPM metrics could be the following: A Key Performance Indicator (KPI) indicates that the duration of the Monitoring and Assessment activities should not exceed 35 seconds. In addition, a business goal dictating that the cost of the whole execution of the normal case sub-process should not exceed 50 units for the normal situation and 500 units for the critical case. The first business goal splits into individual goals, requiring the cost of getting a measurement from each one of the sensors does not to exceed 5 units.

SCC layer: As mentioned in the previous layer, the SCC layer comprises the Web services realizing the activities of the traffic management business process. Specifically, there is an one-to-one mapping between activities and the corresponding Web services. Thus, the composite service of the traffic management application incorporates the Monitoring Web service, the Assessment Web service and the Device Configuration Web service.

The Monitoring service requires as input a Heraklion city's area and provides as output to the Assessment service details about a traffic incident in the critical situation, while in a normal situation it provides environmental and traffic measurements, as well as calendar events. These functionalities are offered by specific service operations:

- The *Check for accident operation* takes as input a city area and provides the location and severity of a car accident as identified by the corresponding car accident monitoring system.
- The *GetTemperature operation* takes as input a string value of the area's name and provides the real value of the current temperature, as identified by the corresponding temperature sensor.
- The *GetHumidity operation* takes as input a city area and provides the float value of the current absolute humidity, as identified by the corresponding humidity sensor.
- The *GetNoise operation* takes as input a city area and provides the float value

of noise measurement, as identified by the corresponding sensor sensor.

- The *GetNO2 operation* takes as input a city area and provides the float value of the current nitrogen dioxide measurement, as identified by the corresponding NO_2 sensor.

- The *GetSO2 operation* takes as input a city area and provides the float value of the current sulfur dioxide measurement, as identified by the corresponding SO_2 sensor.

- The *GetPM10 operation* takes as input a city area and provides the float value of the current measurement of particulates of 10 micrometers or less, as identified by the corresponding PM_{10} sensor.

- The *GetOzone operation* takes as input a city area and provides the float value of the current ozone measurement, as identified by the corresponding ozone sensor.

- The *Measure Traffic operation* takes as input a city area and provides the integer value of the traffic flow measurement, as identified by the corresponding traffic sensor.

- The *Check for Strike days operation* takes as input a city area and informs the Assessment service for strikes in the specific area.

- The *Check for Public Holidays operation* informs the Assessment service for public holidays.

The Assessment service requires as input the monitoring data from the Monitoring service and provides as output the level of actions to be performed by the Device Configuration service. It comprises the following operations:

- The *Assess Incident operation* takes as input the severity and location of the car accident and decides on the actions to be performed by the rescue forces.

- The *Compute AQI operation* takes as input the temperature, NO_2, SO_2, PM_{10} and ozone values and calculates the Air Quality Index (float value).
- The *Compute THI operation* takes as input the temperature and humidity float values and calculates the Temperature-Humidity Index.
- The *Compute ADT operation* takes as input the raw traffic flow measurements during a day and calculates the Average Daily Traffic.
- The *Assess Normal Situation operation* takes as input the values of the previous three operations and decides on the traffic devices' configuration level.

The Device Configuration Service comprises two main operations:

- The *handle traffic incident operation* involves manual activities that are performed by the rescue forces during a critical situation. It takes as input the level of actions and prints the specific actions that were performed.
- The *Device Configuration operation* performs the traffic device Configuration actions. It also takes as input the level of actions and prints the specific actions that were performed.

Furthermore, these three services map to the corresponding SLA documents that are expressed in the WSLA language [Keller and Ludwig 2003]. The Monitoring Service's SLA incorporates the following SLOs:

- *Throughput SLO*: The service throughput for the critical situation should not be lower than 10 ops/sec, while for the normal situation it should not be lower than 4 ops/sec.
- *Execution time SLO*: The service execution time should not exceed 6 seconds in the critical case, and 15 seconds in the normal case.
- *Availability SLO*: The availability of the monitoring service should be greater than 99,99% in critical cases and 99% in normal cases.

The Assessment Service's SLA incorporates the following SLOs:

- *Throughput SLO*: The service throughput for the critical situation should not be lower than 3 ops/min, while for the normal situation it should not be lower than 6 ops/min.
- *Execution time SLO*: The assessment of critical cases should be completed within 20 seconds and within 10 seconds in normal cases.
- *Availability SLO*: The availability of the Assessment service should be greater than 99,99% in critical cases and 99% in normal cases.

Finally, the SLA of the Device Configuration Service incorporates the following SLOs:

- *Throughput SLO*: The service throughput for the critical situation should not be lower than 2 ops/hour, while for the normal situation it should not be lower than 6 ops/min.
- *Execution time SLO*: The handling of critical cases should be completed within 30 minutes, as it requires manual activities and within 10 seconds in normal cases.
- *Availability SLO*: The availability of the Device Configuration service should be greater than 99,999% in critical cases and 99,9% in normal cases.

Software layer: This layer defines the software products required by the individual application services. For the traffic management application, the Assessment service uses the *Drools Rule engine* to decide on the level of actions to be performed, as well as *KairosDB Time-series database* to store the monitoring data and to get aggregated values on user-defined intervals. The Drools rule engine is also utilized by the Device Configuration Service to decide on the concrete actions to be performed based on the assessment.

PaaS and Iaas layers: The deployment of the traffic management application is based on a mixed infrastructure system. More specifically, the Monitoring and

Device Configuration services are deployed on a "medium" Flexiant[3] public VM (2GB RAM, 2-core CPU and 20GB disk), while the Assessment service is deployed on "high" Flexiant VM (4GB RAM, 4-core CPU and 40GB disk), through a PaaS service provided by CloudBees[4] PaaS provider.Each machine provides the required software for the deployed Web services, as well as an Apache Tomcat Application server for hosting the applications. In addition, Flexiant guarantees to the application provider in a Cloud SLA, that the availability of the VM, employed for the deployment of the Monitoring service, will not fall below 99,9%.

Finally, Figure 2.3 summarizes the components description of the traffic management application, depicting a snapshot of the involved components in each layer and their interrelationships.

2.3 Traffic Management Scenario

In this section we motivate our approach with a specific scenario of the traffic management example deployed on multiple Clouds. This SBA, as already analyzed, is characterized by strong interdependencies between the layers and raises various events during its lifecycle [Zeginis et al. 2012b]. The main goal of this scenario is to regulate traffic aiming to optimize particular environmental conditions (e.g., CO_2 levels, temperature and air pollution) drawn from real-time sensor measurements and to properly address car accidents or other incidents impeding normal car flow. Each of these cases is handled by a different sub-process [Zeginis et al. 2012b]. In a normal traffic scenario the three aforementioned tasks take place, while a separate task (T_C) is defined checking for public events and high traffic hours. Each task is a separate software service (SaaS) hosted on underlying PaaS, offering various add-ons (e.g. application server, database, special software, etc.) and IaaS resources, including Compute, Storage and Network components and providing the necessary monitoring sensors.

[3]http://www.flexiant.com/

[4]http://www.cloudbees.com/

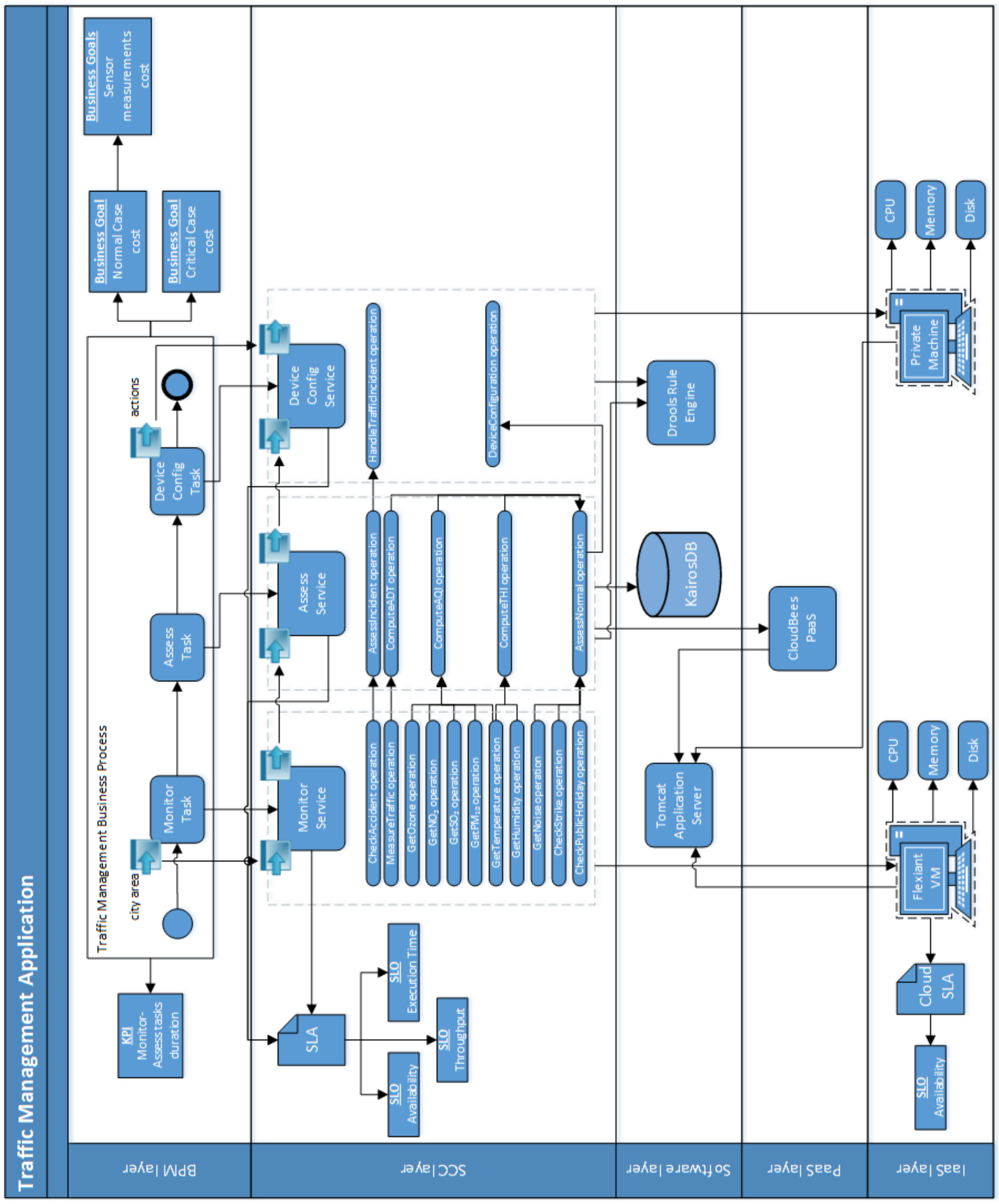


Figure 2.3: The traffic management application components

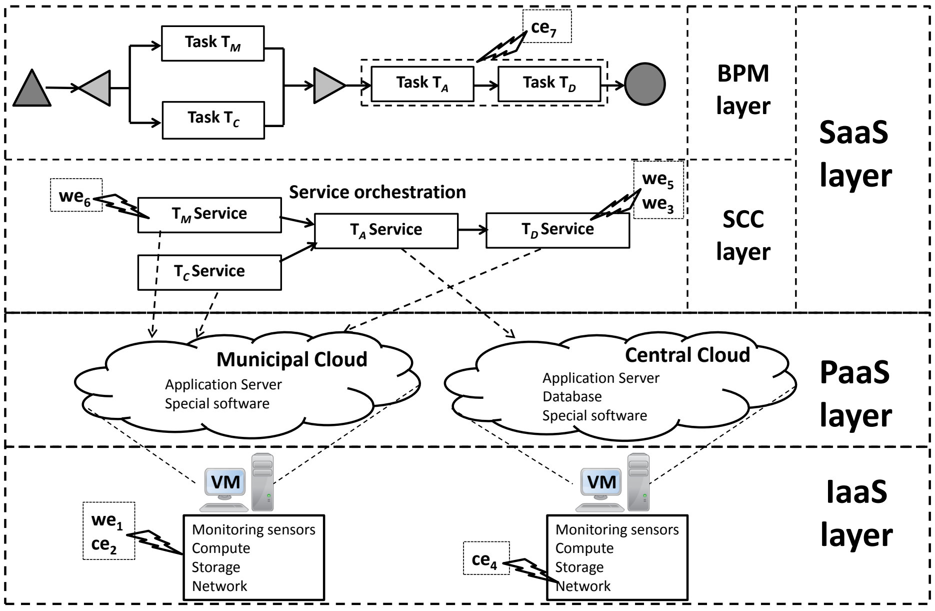


Figure 2.4: Detected violations in the traffic management system

The Cloud infrastructure supporting this SBA includes the following event sources with the particular metric/parameter information in parentheses: (i) the infrastructure, comprising virtualized resources (CPU, memory, disk) and a set of environmental sensors (e.g. temperature, humidity, etc.); (ii) the PaaS service which is used to host the service, or an application server/container which runs on the same VM (e.g. PaaS availability/uptime, response time); and (iii) the hosted application/Web service (e.g. service response and execution time, availability, etc.). The SBA is driven by a business process describing the application tasks, the roles associated with it and the control and data flow (Figure 2.4). Thus, BPM events (e.g., key performance indicators (KPI), business goal) and SCC events (e.g., service response and execution time, throughput, etc.) may be captured at the SaaS layer. The main non-functional goal of the application is to capture the following metrics and interrelate their measurements in order to discover valid event patterns leading to SLO violations:

e_1 $\rightarrow$ Free memory of the VM hosting the Monitor and Device Configuration services

e_2 $\rightarrow$ CPU-load of the VM hosting the Monitor and Device Configuration services

e_3 $\rightarrow$ Execution time of the Device Configuration Service

e_4 $\rightarrow$ Network uptime for the Assessment Service

e_5 $\rightarrow$ Throughput of Device Configuration service

e_6 $\rightarrow$ Average Execution time of the Monitoring service

e_7 $\rightarrow$ Duration of the Assessment and Device Configuration tasks

2.4 Monitoring and Adaptation Scenarios

This section details two monitoring and adaptation scenarios of the traffic management application, which involve a couple of monitoring events detected by the monitoring mechanisms, incorporated in the proposed framework. These scenarios focus on the reactive and proactive adaptation capabilities of the proposed framework and involve pattern discovery and rule derivation issues, in order to exemplify the usage of the proposed algorithms and techniques, analyzed in Chapters 6 and 7.

Reactive adaptation scenario: The response time of the Assessment Service, hosted on an OpenStack[5] VM, is violated. Consequently, the Adaptation Manager triggers a "scaling-out" action to deploy two more "tiny" VM instances with RHEL OS, in order to address the demanding load. This action also requires a "redo activity" action at the BPM layer as the BPEL engine has continued after the triggering of the Assessment activity.

Proactive adaptation scenario: During the application execution, the Monitoring Engine installed on the Flexiant VM hosting the Monitoring service detects that the CPU usage has exceeded the warning threshold of 80%, as identified by the application requirements, while the next execution reports a critical alert of the VM free memory. These two successive warning events have already been related together as a pattern leading to a violation of the execution time SLO of the

[5]http://www.openstack.org

Monitoring service. Consequently, the adaptation engine proactively adapts the system and triggers an elasticity rule dictating a "scaling-up" adaptation action of the VM hosting this service, with the appropriate configuration (i.e. provisioning one more instance of a "medium" Openstack VM with Fedora OS). This action also entails a migration of the Monitoring service to the new provisioned, more powerful VM, offering higher computational power and thus prevents a very possible SLA violation and the underlying imposed penalty.

2.5 Conclusions

To sum up, this chapter provides details about the traffic management running example, used throughout this dissertation to illustrate the applicability of the various components of the proposed monitoring and adaptation framework. The two adaptation scenarios analyzed in the last section of this chapter are reused for the overall evaluation of the framework. The next chapter provides a detailed literature review.

Chapter 3

Literature Review & Fundamentals

Contents

Based on the requirements set in Section 1.4, this chapter provides a literature review of the current state-of-the-art research approaches in Web service monitoring and adaptation. The main purpose of this chapter is to prove that there is a significant research gap in these research areas, paving the way to the next chapters analyzing the thesis' contributions towards closing this gap. In addition, it provides the technical fundamentals for the proposed framework.

This chapter is organized into four main sections. Each section groups the state of the art for each one of the following topics and compares them according to the identified requirements. Specifically, Section 3.1 analyzes the work performed in modeling Cloud application components, while Section 3.2 discusses research approaches for monitoring and adapting SBAs and especially the ones considering cross-layer and multi-cloud aspects. The research approaches are compared against the requirements set in Section 1.4. Approaches offering pattern discovery techniques, utilized by our monitoring framework, are analyzed in Section 3.3. Finally, Section 3.4 aims to equip the interested reader with the appropriate knowledge in order to be able to (i) follow up with the remaining chapters of this dissertation; (ii) fix the terminology used throughout this dissertation in order to make it self-contained; and (iii) briefly analyze the syntax and semantics of the main exploited languages and techniques.

3.1 Cloud and SBA Modeling

As far as the modeling of the business processes of a SBA is concerned, the de facto standard utilized by the whole SOA community is the BPMN 2.0[1] notation, which is a standard for business process modeling that provides a graphical notation for specifying business processes in a Business Process Diagram. This graphical notation facilitates the understanding of the performance collaborations and business transactions between organizations. The primary goal of BPMN is to provide a standard notation readily understandable by all business stakeholders. BPMN 2.0 serves as a common language, bridging the communication gap that frequently occurs between business process design and implementation.

As far as the modeling of Cloud components is concerned, there are some well-

[1]http://www.bpmn.org/

established approaches focusing on the most generic components of a Cloud application, mainly at the infrastructure layer. This section provides a literature review on the most important and widely-used models. In addition, subsection 3.1.1 compares them and pinpoint their deficiencies.

The first attempt to establish a standard for IaaS modeling has been performed by Open Cloud Computing Interface (OCCI)[2]. OCCI, which was originally initiated to create a remote management API for IaaS model based services, has since evolved into a flexible API with a strong focus on integration, portability, interoperability and innovation, while still offering a high degree of extensibility. The main classes in this model are *Resource* and *Link*. The former one describes the main resources of a VM Instance, i.e. Network, Compute and Storage, while the latter one allows the linkage between the previously defined resource types.

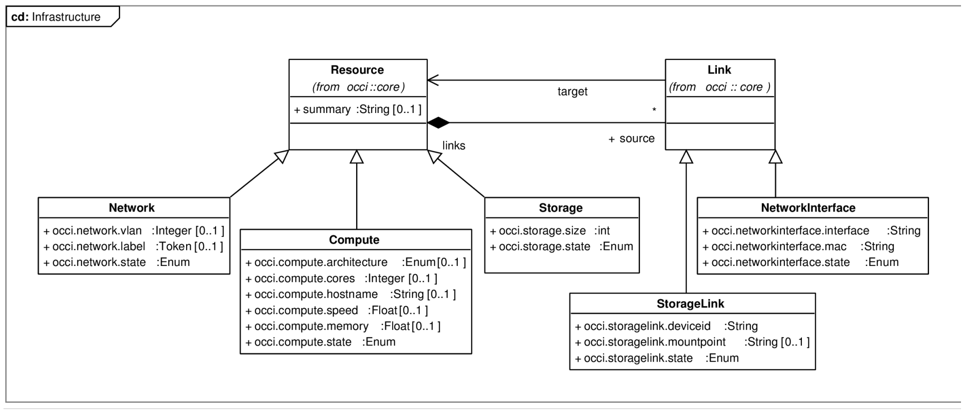

cd: Infrastructure
Resource
(from occi ::core)
+ summary :String [0..1]
target
*
links
+ source
Link
(from occi :: core)
Network
+ occi.network.vlan :Integer [0..1]
+ occi.network.label :Token [0..1]
+ occi.network.state :Enum
Compute
+ occi.compute.architecture :Enum [0..1]
+ occi.compute.cores :Integer [0..1]
+ occi.compute.hostname :String [0..1]
+ occi.compute.speed :Float [0..1]
+ occi.compute.memory :Float [0..1]
+ occi.compute.state :Enum
Storage
+ occi.storage.size :int
+ occi.storage.state :Enum
NetworkInterface
+ occi.networkinterface.interface :String
+ occi.networkinterface.mac :String
+ occi.networkinterface.state :Enum
StorageLink
+ occi.storagelink.deviceid :String
+ occi.storagelink.mountpoint :String [0..1]
+ occi.storagelink.state :Enum

Figure 3.1: Overview of OCCI infrastructure types [Metsch and Edmonds 2011]

OASIS in [Durand et al. 2014] attempts to standardize the PaaS components modeling through the Cloud Application Management for Platforms (CAMP) specification, which describes the artifacts and APIs that need to be offered by a Platform as a Service (PaaS) Cloud to manage the building, running, administration, monitoring and patching of applications in the Cloud. The main purpose of this model (figure 3.2) is to define the artifacts and Application Programming Interfaces (APIs) needed by a PaaS to manage the building, running, administration,

[2]http://occi-wg.org/

monitoring and patching of applications in the Cloud, as well as to enable interoperability among self-service interfaces to PaaS Clouds. This model also enables Cloud vendors to develop new PaaS offerings that will interact with independently developed tools and components.

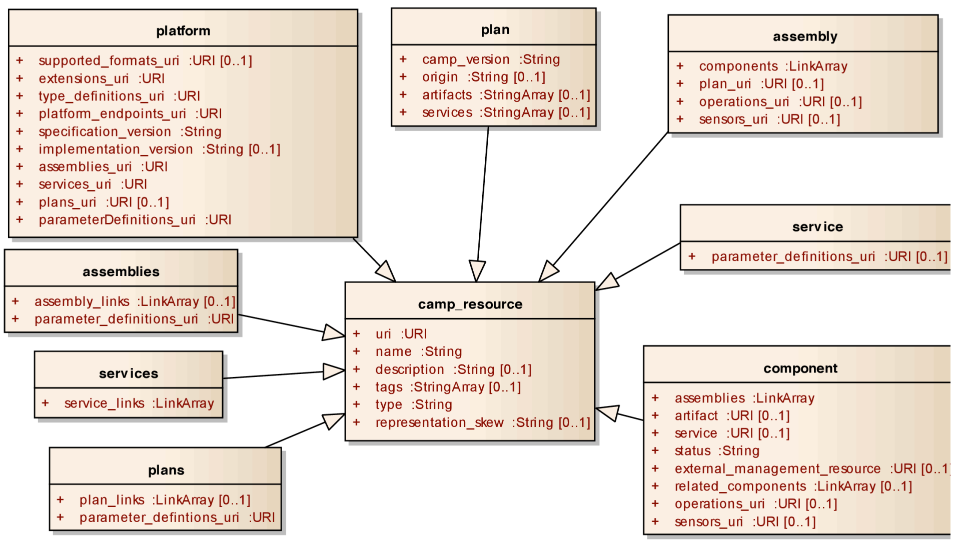

platform
+ supported_formats_uri :URI [0..1]
+ extensions_uri :URI
+ type_definitions_uri :URI
+ platform_endpoints_uri :URI
+ specification_version :String
+ implementation_version :String [0..1]
+ assemblies_uri :URI
+ services_uri :URI
+ plans_uri :URI [0..1]
+ parameterDefinitions_uri :URI
plan
+ camp_version :String
+ origin :String [0..1]
+ artifacts :StringArray [0..1]
+ services :StringArray [0..1]
assembly
+ components :LinkArray
+ plan_uri :URI [0..1]
+ operations_uri :URI [0..1]
+ sensors_uri :URI [0..1]
service
+ parameter_definitions_uri :URI [0..1]
assemblies
+ assembly_links :LinkArray [0..1]
+ parameter_definitions_uri :URI
camp_resource
+ uri :URI
+ name :String
+ description :String [0..1]
+ tags :StringArray [0..1]
+ type :String
+ representation_skew :String [0..1]
services
+ service_links :LinkArray
component
+ assemblies :LinkArray
+ artifact :URI [0..1]
+ service :URI [0..1]
+ status :String
+ external_management_resource :URI [0..1
+ related_components :LinkArray [0..1]
+ operations_uri :URI [0..1]
+ sensors_uri :URI [0..1]
plans
+ plan_links :LinkArray [0..1]
+ parameter_defintions_uri :URI

Figure 3.2: The CAMP PaaS model

Furthermore, OASIS recently standardized Topology and Orchestration Specification for Cloud Applications (TOSCA)[3], in an attempt to enhancing the portability and management of Cloud applications and services across their lifecycle. It is basically a topology of cloud-based Web services, their components, relationships and the processes that manage them. The TOSCA standard includes specifications to describe processes that create or modify web services. Although, it is the latest related attempt, it has attracted the interest of many researchers along with Cloud Modeling Language (CloudML)[4] (they are actually starting to converge) when it comes to modeling Cloud deployments.

One of the results of Cloud4SOA EU Project[5] was the modeling of the PaaS com-

[3]https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[4]http://cloudml.org
[5]http://www.cloud4soa.eu/

ponents[6]. Through this model, PaaS providers can semantically annotate their PaaS offerings. Specifically, it captures the available platform functionality as well as technical- and business-driven characteristics of PaaS offerings. Another attempt to model Cloud computing entities has been proposed within mOSAIC EU Project[7]. The proposed ontology [Moscato et al. 2011] aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by users. This platform can be used to improve interoperability among existing Cloud Solutions, platforms and services.

Finally, the modeling at the Software-as-a-Service (SaaS) layer has been widely influenced by the SOA, providing plenty SBAs. Within the S-Cube EU project[8], specific models, both for the SCC and the BPM layers have been proposed. These models define the main concepts of a Business Process (i.e. activities, roles, KPIs) and the corresponding composite services (simple services, constraints, SLAs), as well

[6]http://islab.uom.gr/cloud4soa/c4smodel.html/ (last access 26/9/2014)

[7]http://www.mosaic-cloud.eu/

[8]http://www.s-cube-network.eu/

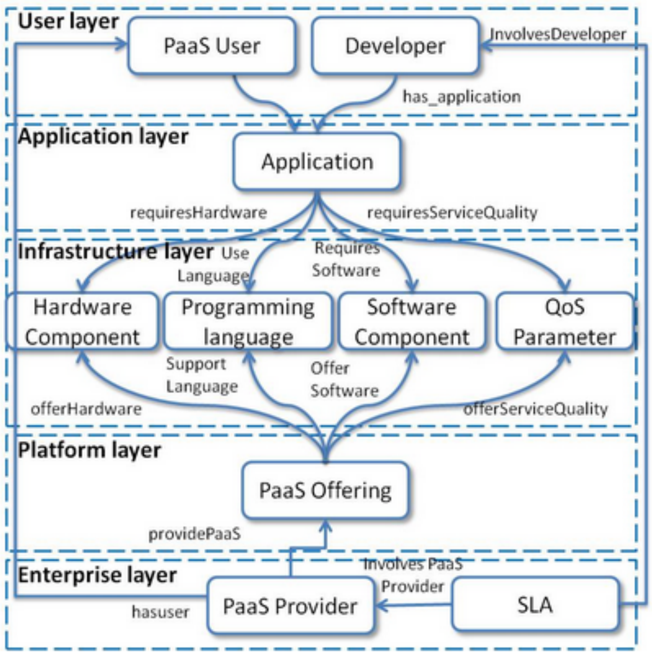


Figure 3.3: The Cloud4SOA PaaS model

as their interrelationships (e.g. a service realizes an activity).

Concerning modeling of the interface incompatibilities between services of an orchestration, the authors in [Taher et al. 2013] propose and define a meta-model which has as main objective the modeling of the provider and client service interfaces, the incompatibility patterns and the adapters for solving both signature and protocol incompatibility problems that may exist between Web service protocols. Finallt, the authors in [Inzinger et al. 2013] propose an approach for model-based adaptation of cloud computing applications, supporting the SBA customers by providing a simple SBA model on which they can specify the desired application behavior.

3.1.1 Comparison

The aforementioned approaches do not offer a unique unified modeling solution, describing all the components of a multi-cloud SBA. Most of these models are layer specific, providing only high-level details for the other layers and none of them combines information for components from the SOA and Cloud context. Moreover, the attributes they provide lack information about the adaptation process that can facilitate a rule expert to define suitable adaptation strategies to be applied on each of the modeled components. Thus, there is a need for a holistic model describing all the components of a multi-cloud SBA, considering both the SOA (i.e., BPM, SCC and SI) and Cloud layers (i.e., SaaS, PaaS and IaaS). Additionally, a meta-model of the adaptation actions that can be applied on each one of the system components, will profoundly assist the mapping process from monitoring events to suitable adaptation actions.

3.2 Monitoring and Adaptation

The need for monitoring different functional and non-functional requirements, as well as for taking adaptation actions is widely recognized by industry and academia, as a means of improving SBAs. In recent years, a couple of approaches towards monitoring and adaptation of SBAs have been proposed. The aim of this section is to analyze these approaches, especially the ones featuring cross-layer (subsection 3.2.1) and Cloud (subsection 3.2.2) aspects, and present their main drawbacks. Subsection 3.2.3 compares and evaluates these approaches based on the identified requirements of Section 1.4 and draws conclusions.

[Baresi et al. 2007] presents an approach for self-healing of BPEL processes. This approach is based on Dynamo [Baresi and Guinea 2005] monitoring framework along with an AOP extension of ActiveBPEL and a monitoring and recovery subsystem using Drools Event-Condition-Action (ECA) rules. A composition designer provides assertions for invoking, receiving or picking activities in the business process. These assertions can be specified using two domain specific languages (WSCoL and WSReL). The problem of selecting alternative services and dealing with possible interface mismatches when forwarding a request to an alternative endpoint recovery is not explicitly addressed. Additionally, the recovery rules cannot be changed dynamically, as they need to be compiled offline.

The VieDAME environment [Moser et al. 2008] extends the ActiveBPEL engine to enable BPEL process monitoring and partner service substitution based on various strategies. The services are selected according to defined selectors. VieDAME requires service registration to a repository, marking services to be monitored and eventually substituted as replaceable. It uses an engine adapter to extend the engine's functionality, but does not explicitly address fault handling.

[Barbon et al. 2006] presents an event-based monitoring approach, developed within the Astro project, which also extends the ActiveBPEL engine and defines RTML, an executable monitoring language to specify SBA properties. Events are combined by exploiting past-time temporal logics and statistical functions. Monitors run in parallel with the BPEL process as independent software modules verifying the guarantee terms by intercepting the input or output messages received or sent by the process. This work does not allow for dynamic (re-)configuration of the monitoring system in terms of rules and meta-level parameters.

In [Mahbub and Spanoudakis 2007, Spanoudakis and Mahbub 2006] the authors present an approach towards extending WS-Agreement [Andrieux et al. 2007]. This approach supports monitoring of functional and non-functional properties. EC-Assertion is introduced to specify service guarantees in terms of different types of events, which are defined in a separate XML schema and it is based on Event Calculus (EC) [Shanahan 1999]. By proceeding in parallel with the business process execution, it leads not only to less impact on performance, but also to a smaller degree of responsiveness in discovering erroneous situations.

[Farrell et al. 2004] presents an SLA-based monitoring approach exploiting EC as the underlying formalism. This approach addresses the utility computing domain, where the cornerstone aspect is to provide resources with certain quality characteristics. Contracts are defined based on contract patterns, which are then axiomatized using EC. Then, the effects of critical events on the contract state/evolution are defined. The respective framework is based on a particular architecture and comprises an analyzer managing the contract analysis and reporting and a visualizer representing the SLA monitoring results.

A platform for developing, deploying and executing SBAs is proposed in [Curbera et al. 2005], incorporating tools and facilities for checking, monitoring and enforcing service requirements expressed in WS-Policy notations. The Colombo platform comes with a module that manages policy assertions. Apart from evaluating the assertions attached to particular service-related entities at both design and run-time phases, the framework provides the means for policy enforcement, e.g., it may approve a delivery of a message, a rejection of it, or defer further processing.

The authors in [Inzinger et al. 2014] introduce an architecture and a DSL, named MONINA (Monitoring, Integration, Adaptation), that allow to integrate functionality provided by different components and to define monitoring and adaptation functionality. It is similar to our approach, as monitoring is carried out by complex-event processing queries, while adaptation is performed by condition action rules performed. However, it differentiates regarding its scope, which aims at the specification of platforms integrated into a Virtual Service Platform (VSP), that provides a unified view on the functionality of the integrated service platforms that are connected by control interfaces. In addition, it lacks cross-layer and multi-cloud features, as well as experimental analysis of the implemented approach.

3.2.1 Cross-layer Approaches

[Kazhamiakin et al. 2009a] is the first approach that reveals the need for cross-layer monitoring and adaptation of SBAs. It can be considered as the stepping stone of all the following approaches. In particular, this work defines a set of requirements for a novel and integrated approach that provides a coherent and holistic solution for monitoring and adapting the whole SBA. It illustrates the problem using a series of case studies and provides a set of requirements that a novel cross-layer approach should address. Based on the taxonomy of those requirements, the authors define the mechanisms and techniques that are necessary for addressing the requirements and that constitute an integrated cross-layer framework. Finally, it describes a uniform conceptual model underlying such a framework and presents a set of potential and existing principles and methodologies that enable cross-layer monitoring and adaptation. In this work Kazhamiakin et al. made a first attempt to introduce the fundamental principles for cross-layer monitoring and adaptation of Service-Based Applications. Although this work does not provide a concrete solution to the problem, it offers helpful directions for aspiring researchers in this area. Therefore, this work cannot be compared to other research work proposing specific frameworks, but it can be considered as a benchmark, as it provides the characteristics of an ideal cross-layer monitoring and adaptation system.

In [Popescu et al. 2012; 2010] the authors propose a methodology for the dynamic and flexible adaptation of multi-layer applications using adaptation templates and taxonomies of adaptation mismatches. Templates are exposed as executable BPEL processes that may encapsulate adaptation techniques. The template developers are in charge of associating the templates they develop with adaptation mismatches based on the types of mismatches they can cope with. For each application layer, one may define one or more taxonomies of adaptation mismatches, which may either be generic or contain domain information for particular application domains. The authors use tree-based taxonomies and *is-a* relationships between children and parent mismatches, as well as for the scaled degree of matching between adaptation mismatches. The cross-layer dimension of this approach is achieved by linking adaptation templates, corresponding to layers where adaptation is needed, either directly or indirectly. In the former case, a BPEL adaptation template invokes the WSDL interface of another BPEL adaptation template. In the latter case, a BPEL adaptation template raises an event that will trigger the selection, deployment and execution of another adaptation template. This can be achieved by using standard BPEL activities that are invoked to generate events and receive or pick branches to receive events. Within each layer the authors assume the availability of several adaptation templates, some of which are linked and which are associated with different taxonomy mismatches.

This work deals with cross-layer adaptation in a flexible and dynamic manner. It assumes that monitoring takes place before this procedure without going into details about monitoring events are collected from different layers. The proposed layers correspond to BPM and SCC layers. Its main drawback is that it does not take into consideration infrastructure failures, which usually occur during the execution of SBAs.

In [Guinea et al. 2011] the authors present an integrated approach for monitoring and adapting multi-layered SBAs. This approach is based on a variant of MAPE control loops [Horn 2001] that are typical found in autonomic systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that they always reason holistically and adapt the system in a cross-layered and coordinated way. The proposed methodology comprises four main steps: (i) Monitoring and Correlation, where sensors capture run-time data about the software and infrastructural elements, (ii) Analysis of Adaptation Needs, in which the framework identifies anomalous situations and pinpoints where it needs to adapt, (iii) Identification of Multi-layer Adaptation Strategies, in which the framework uses the adaptation capabilities that exist within the system to define a multi-layer adaptation strategy as a set of software and/or infrastructure adaptation actions; and (iv) Adaptation Enactment, where different adaptation engines at the software and infrastructure layer enact their corresponding parts of the multi-layer strategy. This approach comprises a set of mechanisms to provide multi-layer monitoring and adaptation. Its main drawback is that it does not feature proactive adaptation capabilities. In addition, it does not provide in detail how cross-layer monitoring is performed in which the various events are synchronized.

Another cross-layer adaptation approach is presented in [Gjørven et al. 2008], where the authors propose an approach towards cross-layer self-adaptation, which exploits mechanisms across two layers: (i) the Service Interface layer; and (ii) the Application layer, related to the BPM and SCC layers respectively. In the Service Interface layer, loosely coupled services communicate via open protocols, hiding their implementation and technology platform, whereas at the Application layer, service application logic is developed and deployed on different technology platforms. A technologically independent middleware, named QUA is introduced to support coordinated cross-layer adaptation. This middleware consists of a planning framework, which is responsible for selecting service implementations and configurations, a platform framework, which encapsulates mechanisms for managing and adapting services and a supporting Meta-Object Protocol Service. This work is a coarse-grained approach, that adopts only the aforementioned layers. The proposed technology-agnostic adaptation middleware can be used to integrate and exploit adaptation techniques and mechanisms from both application and service layers. The authors do not elaborate on cross-layer monitoring techniques. Finally, this framework does not have any proactive adaptation capabilities.

[Zengin et al. 2011] proposes a holistic SBA management framework, called CLAM, which can deal with cross- and multi- layer adaptation problems. This is achieved in two ways. On the one hand CLAM identifies the application capabilities affected by the adaptation actions and on the other hand it identifies an adaptation strategy that solves the adaptation problem by properly coordinating a set of specific adaptation capabilities. This work addresses the cross-layer adaptation problem. The tree-based approach for defining adaptation paths seems very interesting although it can be time-consuming. In addition, during the ranking process of the adaptation branches, cost is not taken into consideration. A drawback of this approach is that it does not elaborate on cross-layer monitoring. Finally, this approach has neither proactive adaptation, nor functional aspects handling capabilities.

In [Jiang et al. 2011], the authors present the design and implementation of an experimental facility for cross-layer adaptation. It is based on adaptation logic that builds run-time adaptation models based on information from the application, communication and hardware layers and uses the model for coordinated adaptation of these layers. The work builds on the MUSIC [Rouvoy et al. 2009] adaptation framework, based on an externalized approach to self-adaptation where the adaptation logic is delegated to generic middleware. This approach addresses cross-layer adaptation from a different viewpoint than the previous ones. It focuses on the testing and evaluation of distributed systems with regard to their adaptive behavior. Unfortunately, the current implementation supports only events from the communication layer, but there are plans for other types of simulation. To sum up, it is a very interesting approach that we could exploit for evaluating the proposed/envisioned adaptation framework.

[Schmieders et al. 2011] presents a solution to avoid SLA violations in the complex settings of SBAs. The novelty of the approach is the exploitation of all SBA layers for the prevention of SLA violations. The authors propose a framework that integrates layer specific monitoring and adaptation techniques and enables multi-layered control loops in service-based systems. The identification of adaptation needs is based on SLA prediction, which uses assumptions on the characteristics of the running execution context. Multiple adaptation mechanisms are available to react on the adaptation need, acting on different layers of the SBA. The adaptation strategy chooses the right adaptation mechanism, coordinated by a multi-agent community. The proposed approach efficiently addresses cross-layer adaptation by preventing SLA violations. An interesting perspective of this work is that it implements SALMon monitoring mechanism, as an SBA by itself, in order to be easily integrable with other technologies in the framework. Its main limitations are that it does not comprise infrastructure monitoring unlike SI adaptation and that the proposed adaptation process is not proactive.

[Fugini and Siadat 2009] proposes a SLA contract that merges parameters from user, business service and IT infrastructure as an alternative approach for the cross-layer monitoring and adaptation of SBAs. The user parameters are measured through the Key Goal Indicators (KGIs) that define how well service-based processes achieve the customers' goals. Once the contract is created, it is evaluated in the monitoring phase and checked for possible violations from the predefined values. According to the source of a violation event, an appropriate action is taken to adapt the violated condition to the new values, through an adaptation mechanism. In the last phase of the proposed framework, the contract is updated to fulfil the new conditions and requirements. The proposed approach efficiently combines parameters from user, business services and IT infrastructure to construct SLA contracts. These contracts can be exploited by existing approaches to determine the monitoring events and the corresponding adaptation actions at each layer. An innovative aspect is the idea of using user profiles in the contract, to express interests on data and parameters as numerical scores or explicit ordering relations, thus making it more personalized. Finally, a limitation of this work is that only non-functional characteristics of SBAs can be defined in such types of contracts.

[Leitner et al. 2010a] introduces the PREvent framework, that utilizes machine learning techniques to predict SLA violations. An event-based monitoring system [Michlmayr et al. 2010] feeds the adaptation engine, which automatically prevents these violations by predicting them, using regression from monitored run-time data and applying adaptation actions in service composition at run-time. In particular, it focuses on endangered composition instances and does not apply on the SBA class. Its adaptation capabilities are limited in comparison to other aspect-oriented programming (AOP) approaches [Karastoyanova and Leymann 2009, Kongdenfha et al. 2009, Leitner et al. 2010b], that also employ structural adaptation on service compositions. Its evaluation reveals satisfactory results in predicting and preventing most of the violations.

In [Hielscher et al. 2008] the authors propose PROSA framework, that enables proactive adaptation by exploiting online testing techniques to detect changes and deviations before they actually lead to undesired consequences. Oppositely to offline testing, which is performed during the design phase, online testing is done during the operation phase of a SBA and thus does not affect its execution. The main benefit of this approach is that it enables both reactive and proactive adaptation, as well as it can select the optimal adaptation decision through "pre-testing" of the alternatives, while a pinpointed drawback is the lack of its applicability demonstration in real-world systems, as well as the impact of the execution of test cases on the performance of the SBA.

The work presented in [Kazhamiakin et al. 2009b] introduces an adaptation approach for SBAs, based on a process quality factor analysis [Wetzstein et al. 2009]. The result of this analysis is a decision tree, called a dependency tree as it shows the dependencies of KPIs on process quality factors from different functional levels of an SBA. These dependency trees are also used to model adaptation actions and associate them with quality metrics, as well as to extract adaptation requirements from the dependency tree and come up with an adaptation strategy.

On the same wavelength, [Dranidis et al. 2010] proposes an approach for "just-in-time" (i.e., shortly before a conversational service is invoked for the first time) proactive adaptation, utilizing test cases grounded in the formal theory of Stream X-machines (SXMs) [Dranidis et al. 2007]. Most of the costly test preparation activities are executed during deployment time, in order not to interfere with the context of the concrete SBA. The applicability of the approach is illustrated on a e-shop example, revealing that online tests should be limited not only to economic and technical issues, but also to the impact they have on the actual SBA performance.

Moreover, the authors in [Song et al. 2013] propose a model-based approach towards cross-layer system monitoring and adaptation. In particular, they propose meta-modeling languages for system experts to specify the layers, the relations between them, as well as the constraints on each layers. However, this approach is an initial attempt, that has not been sufficiently evaluated on real life SBAs. Moreover, this approach is generic, applicable on distributed multi-tier systems, not providing concrete features for SOA and Cloud architectures.

Finally, in [Dib et al. 2012] the authors make two contributions regarding dynamic adaptability of modern operating systems . First, they propose a service-oriented approach for building distributed OSs, which enables adapting the OS algorithms in a dynamic, on-demand fashion. Second, they propose a common framework to adapt both the OS and application layer in a coordinated way, thus avoiding any possible conflict or redundancy. Consequently, this approach is a cross-layer one in the sense that it supports both the OS and application layer. Nevertheless, it does not focus on the layers of the SOA and/or Cloud stack.

3.2.2 Cloud Monitoring and Adaptation

While several Cloud monitoring approaches have been proposed in the past, only a few take comprehensively into account cross-layer issues. In what follows we will first describe the most prominent of such approaches and then compare them in the next subsection 3.2.3 according to the requirements established in Section 1.4.

One of the main barriers to the adoption of SBAs is the concern raised over the trustworthiness and reliability of third-party services utilized in a SBA. The problem of reliability becomes more complex when third-party Cloud computing services are used as the underlying infrastructure for provisioning the SBA. [Bratanis et al. 2011] focuses on the definition of SLAs in the BPM, SCC and SI layers, for cross-layer adaptation and monitoring of SBAs. They offer insights into how an SBA should be analyzed, in order to identify and separate the distinct business, software and infrastructure services. This technique is applied on a case study that concerns PaaS offerings for enabling the customization of SaaS applications by third parties. The presented case study clearly states the service characteristics and the SLOs at each SBA layer. Its main drawbacks are that it does not investigate methods for representing layer-specific SLAs and that there is no cross-layer implementation to apply these SLAs on.

[Alcaraz Calero et al. 2012] presents an analysis of a wide set of distributed monitoring solutions analyzing the features, requirements and topology of a cross-layer monitoring system for Cloud computing. Similarly, [Hasselmeyer et al. 2012] provides an overview of monitoring in Cloud environments, highlighting the similarities with grid monitoring solutions. The authors pinpoint the requirements for each one of the involved roles and present a solution for interoperable and vendor-independent Cloud monitoring.

Moreover, a number of EU-funded research projects are currently examining Cloud monitoring solutions. IRMOS[9] offers a Cloud infrastructure, comprising a service management system, that acts as a link between SaaS and IaaS to manage the negotiation, reservation, execution and monitoring of the Application Service Components and, at the same time, to check at run-time performance related requirements while conforming to the Service Level Agreements (SLAs). RESERVOIR[10] introduces Lattice, a non-intrusive monitoring framework for Cloud applications. Lattice features probes for collecting and transmitting data to the service management part, which takes decisions based on them. VISION Cloud[11] proposes

[9]http://www.irmosproject.eu/
[10]http://www.reservoir-fp7.eu
[11]http://www.visioncloud.eu/

a monitoring framework able to aggregate events, apply rules on them and generate new events, representing complex states of the system. Cloud4SOA[12] proposes a cross-PaaS management and monitoring system [Zeginis et al. 2013b] for applications hosted on multiple Clouds, in order to ensure that their performance consistently meets expectations and that Cloud resources are being effectively utilized, despite the technology used to implement such applications.

3.2.3 Comparison

In the previous section we analyzed the current approaches on SBA monitoring and adaptation, including cross-layer and Cloud approaches. In this section we compare these approaches based on the set of requirements defined in Section 1.4 and Table 3.1 summarizes this comparison. The ✓ mark indicates satisfaction of the requirement, the ✗ mark defines that the approach does not satisfy the requirement, while the $\sim$ mark shows uncertainty. Regarding the timeliness criterion, *Im* means immediate delivery of the monitoring event and *Pm* postmortem notification for the monitoring events after a single execution of the SBA, *F* and *NF* represents the functional and non-functional monitoring capability of the considered approach, while *I* and *C* the instance and class scope respectively.

From this comparison of current cross-layer approaches we conclude that none of them satisfies all the requirements posed for a complete and holistic cross-layer monitoring and adaptation approach for multi-cloud SBAs. It is worth mentioning, that after this comprehensive literature review, we have concluded that most approaches focus on either monitoring or adaptation, not providing research contribution for both these close related procedures. In particular, as far as dynamicity is concerned, most of these works perform dynamic adaptation unlike dynamic monitoring. In addition, a few of current approaches are intrusive regarding both monitoring and adaptation, while almost all adaptation approaches are reactive rather than proactive. As far as the type of monitored properties is

[12]http://www.cloud4soa.eu/

concerned, the majority of the reviewed approaches addresses only functional attributes, while the scope of most of them is instance-based.

Moreover, current approaches addressing cross-layer monitoring and/or adaptation do not take into account all layers (Cloud and SOA), while the multi-cloud ones do not provide the required flexibility and proactive adaptation capabilities. The main strength of our approach is that it deals with both service and Cloud-based applications, while considering challenges raised in a Multi-Cloud environment, satisfying almost all the considered requirements. Our approach can deal with both functional and non-functional properties; its scope is instance-based; it provides both reactive and proactive capabilities, while it immediately notifies about the detection of the monitoring events.

Table 3.1: Comparison of Cross-layer Monitoring and Adaptation approaches

Approaches	Cross-layer/	Dynamicity		Intrusiveness		Timeliness		Type of properties		Cloud properties			
	Multi-Cloud	Mon.	Adapt.	Mon.	Adapt.	Mon.	Adapt.	Kind	Scope	Multi-tenant	Dyn./Ext.	Scalability	Simplicity
[Kazhamiakin et al. 2009a]	✓ / ✗	∼	∼	∼	∼	∼	∼	∼	∼	✗	✗	✗	✗
[Popescu et al. 2012; 2010]	✓ / ✗	✗	✓	✗	✓	Im	R	F	I	✗	✗	✗	✗
[Guinea et al. 2011]	✓ / ✗	✗	✓	✗	✓	Im	R	NF-F	I-C	✗	✗	✗	✗
[Gjørven et al. 2008]	✓ / ✗	∼	✓	∼	✗	∼	R	F	I	✗	✗	✗	✗
[Zengin et al. 2011]	✓ / ✗	✗	✓	✗	✓	Im	R	NF	I	✗	✗	✗	✗
[Jiang et al. 2011]	✓ / ✗	✗	✓	✗	∼	Im	R	NF	I-C	✗	✗	✗	✗
[Fugini and Siadat 2009]	✓ / ✗	∼	∼	∼	∼	Im	R	F	I	✗	✗	✗	✗
[Bratanis et al. 2011]	✓ / ✗	∼	∼	∼	∼	∼	∼	NF-F	I-C	✗	✓	∼	∼
[Schmieders et al. 2011]	✓ / ✗	∼	∼	✗	✗	Im	R	NF	I-C	✗	✗	✗	✗
[Leitner et al. 2010a]	✗ / ✗	✓	✓	✗	✗	Im	P	NF	I	✗	✗	✗	✗
[Hielscher et al. 2008]	✗ / ✗	∼	✓	∼	✗	∼	P-R	NF-F	I	✗	✗	✗	✗
[Kazhamiakin et al. 2009b]	✓ / ✗	∼	✗	∼	✗	Im	P	NF	I	✗	✗	✗	✗
[Dranidis et al. 2010]	✗ / ✗	∼	∼	✗	✗	Im	R	NF	I-C	✗	✗	✗	✗

[Alcaraz Calero et al. 2012]	✓ / ✗	∼	✗	✗	∼	Im	∼	NF	∼	✓	✓	✓	✓
[Hasselmeyer et al. 2012]	✗ / ✗	✓	✗	∼	∼	Im	R	NF	I	✓	✓	✓	∼
IRMOS project	✗ / ✓	✗	∼	✗	✗	Im	∼	NF	I	✓	∼	∼	✓
RESERVOIR project	✗ / ✓	∼	∼	✗	✗	Im	R	NF	I-C	✓	∼	✓	✓
VISION project	✗ / ✗	∼	∼	✗	✗	Im	∼	NF	I-C	✓	∼	✓	∼
Cloud4SOA project	✗ / ✓	∼	✓	✗	✗	Im	∼	NF	I-C	✓	✗	✓	✓
[Baresi et al. 2007]	✗ / ✗	✓	✗	✓	✓	Pm	R	NF-F	I	✗	✗	✗	✗
[Moser et al. 2008]	✗ / ✗	✗	✓	✗	✗	Im	R	NF	I	✗	✗	✗	✗
[Barbon et al. 2006]	✗ / ✗	✓	✗	✗	✗	Im	✗	NF	I-C	✗	✗	✗	✗
[Mahbub and Spanoudakis 2007]	✗ / ✗	✗	✗	∼	✗	Pm	✗	NF-F	I	✗	✗	✗	✗
[Farrell et al. 2004]	✗ / ✗	∼	✗	✗	✗	Im	✗	NF	I	✗	✗	✗	✗
[Curbera et al. 2005]	✗ / ✗	∼	✗	✗	✗	Pm	✗	NF-F	I	✗	✗	✗	✗
Proposed framework ECMAF	✓ / ✓	✓	✓	✓	✗	Im	R-P	NF-F	I	✓	✓	✓	✓

3.3 Pattern Discovery

This section reviews the related work in pattern discovery, from simple mathematical solutions to more complex logic-based approaches. We distinguish the studied approaches in three main categories: (i) Mathematical and Statistical approaches (Subsection 3.3.1) that predominantly make use of the minimum threshold of an event sequence occurrence, (ii) Temporal approaches (Subsection3.3.2) that introduce temporal constraints to discover patterns in different granularities; and (iii) Logic-based approaches (Subsection 3.3.3) that use the power of logic to discover frequent patterns. The following subsections analyze a series of works based on the aforementioned categorization. Finally, Subsection 3.3.4 highlights the main benefits of each approach and compares them, based on a set of criteria.

3.3.1 Mathematical and Statistical Approaches

This type of approaches introduces methods of discovering frequent event patterns, exploiting a user-defined minimum frequency or support (*minsup*) as mainly used in data mining area. Frequent patterns or *itemsets* are those with greater or at least equal than this *minsup* occurrences.

The springboard of all these approaches is the *Apriori* algorithm, firstly introduced in [Agrawal and Srikant 1994]. This algorithm produces a set of all significant association rules (i.e. rules relating a set of variables) between items in a large database of transactions. The main concept of Apriori algorithm is to discover sets of items that have a *minsup*. In this context a subset of a frequent itemset is also considered a frequent itemset. The algorithm iteratively finds frequent itemsets with incremental cardinality, from 1 to *k* (*k*-itemset) and uses them to generate association rules. A typical example to exemplify the algorithm's application is the market basket, where the consumer's products are associated with each other. For example, an association rule could be the following: $\{flour, sugar\} \Rightarrow \{eggs\}$, provided that the number of consumers buying all these three products surpasses the *minsup*.

In [Römer 2006], the author introduces a method for discovering frequent event patterns, as well as their spatial and temporal properties in sensor networks, exploiting data mining techniques. Provided that events are put into a spatial and temporal context, the proposed system [Römer 2008] correlates certain type of events on a sensor node with context events in a confined neighborhood in the recent past. Thus, a correlation of events (i.e. distributed event patterns) is discovered whenever these patterns' frequency surpasses a *minsup*. The proposed multi-pass algorithm splits the event stream in memory-fitted blocks, finds the frequent patterns in each block using the special minsup $\overline{MS}$ for each block, and then incrementally builds the set of patterns for the whole stream. Furthermore, another minsup-based algorithm is presented in [Zaki and Hsiao 2002] to discover closed itemsets, i.e. an itemset which none of its immediate supersets has the same support as the itemset itself.

In [Manku and Motwani 2002], the authors propose another widely used algorithm, named *Lossy Counting*. This is a one-pass algorithm, which computes approximate frequency counts of elements in a data stream and it involves grouping the row items into blocks or chunks and counting within each chunk. A counter is preserved for each itemset. It periodically discards elements with very low count from the table. In this way, the most frequently accessed itemsets will almost never have low counts, whereas itemsets that do not hit very often may not be tracked anymore as their count would drop to zero in a few blocks. The main benefit of this algorithm is that it requires provably small main memory footprints. Lossy Counting algorithm outmatches the Apriori one, in terms of memory usage, but it requires much more time to find frequent itemsets in a data stream.

Another single-pass algorithm for mining frequent itemsets, named DSM_FI, is proposed in [Li et al. 2004]. This algorithm is based on a windows model and comprises four main steps: (i) reading a set of transactions, (ii) construction of the summary data function, (iii) infrequent information pruning, (iv) top-down traversal of the summary data structure to discover frequent itemsets. The algorithm's evaluation demonstrates that the DSM_FI algorithm significantly outperforms the Lossy Counting in terms of execution time and memory usage.

In [Hellerstein et al. 2002], the authors propose a framework for discovering actionable patterns in large datasets of historical events. The main contribution of this work is that it also takes into consideration the attributes of the events, in order to group and itemize events. Specifically, this work deals with four main types of event patterns, i.e. event bursts, periodic patterns, mutually dependent patterns and multi-attribute patterns. For the first ones, periods with higher event rates are isolated and handled separately, while for the periodic patterns they introduce the concept of partially temporal association related to *p-patterns*, which are discovered by firstly finding the period lengths for each periodic event and then discover its temporal associations. The mutually dependent patterns (defined as *m-patterns*) define itemsets, for which, if a subset appears, the remaining items of the *m-pattern* occur with high probability. This approach is similar to our approach for proactive adaptation. Finally, *maf-patterns* are presented to capture multi-attribute frequent itemsets, which are very efficient for systems' management. This method groups events based on a set of attributes and then mines on these subset of events to discover frequent itemsets.

The authors in [Laxman et al. 2007] introduce a pattern discovery framework utilizing event duration information explicitly defined into the episode structure (i.e., *generalized episodes (GE)*). This temporal information improves the descriptive power of episodes and reveals duration-dependent correlations in the data stream. Thus, the discovery process of *GE*s can then help a root-cause analysis for persistent causative chains of events, the frequency of which is above a given threshold. In particular, the associated formalism for defining *GE*s, as well as the corresponding discovery algorithms are presented. The framework's evaluation reveals significant accurate discovery and ranking results even for overlapping episode.

In [Chaturvedi et al. 2013] a pattern mining tool is presented that can help in improving the efficiency and effectiveness of rule-based data standardization systems, usually employed by business to improve their data quality. The extracted patterns can be further exploited by the domain expert for rule definition. A greedy algorithm is introduced to mine significant semantically relevant sub-patterns, as well as a clustering model to group the discovered sub-patterns which encode the same domain knowledge, through the specification of a special distance measure. The tool also exhibits ranking capabilities, exploiting the normalized frequencies of the mined sub-patterns. Its evaluation reveals its accuracy in discovering all the related sub-patterns coded by rule experts.

[Hamilton-Wright and Stashuk 2008] presents an approach for statistically based pattern discovery (SBPD), that extracts patterns based on a test of statistical validity. Its main objective is to find relations between specific values of features in a biological training dataset. For this reason, the proposed algorithm observes *polythetic events*, i.e., events defined by the occurrence of combination of feature values (e.g., a record of a patient comprises her/his age, gender, disease state, disease duration, etc.). A single feature measurement is considered as a *primary event*, while the co-occurrence of a specific set of primary events forms a *high order event*, the frequent detection of which is considered as a pattern that provides useful knowledge for inference and reasoning. The discovered patterns can be further ranked exploiting their statistical confidence and the use of specific classifiers.

3.3.2 Temporal Approaches

This type of approaches introduces methods of discovering frequent event patterns, exploiting the temporal relations among the events of the input stream. These approaches can be very useful for deriving implicit information for the temporal ordering of the raw data and for predicting the future behavior of the monitored application. They can also consider *minsup* thresholds, but they are examined separately, as they address the pattern discovery problem from a different perspective.

In [Bettini et al. 1998] the authors introduce a formal framework for expressing data mining tasks involving time granularities, as well as algorithms for performing these tasks. In particular, time constraints are injected into the system to bound the distance between a pair of events in terms of time granularity. For instance, event e_2 must happen within two minutes after the occurrence of event e_1 in order to consider e_1, e_2 an event pattern. The notion of *timed finite automaton with granularities (TAG)* is introduced that is a standard automaton, the clocks of which run on different granularities, in order to discover patterns with different frequencies. Moreover, a number of heuristics is used to limit the candidate patterns and optimize the framework's performance. Evaluation results on a stock trading dataset show its significant performance on mining frequent pattern with various granularities, especially with the exploitation of heuristics, reducing the the candidate event types.

The authors in [Patnaik et al. 2012] present a temporal data mining approach for data that cannot fit in memory or for data being processed at a faster rate than the generation one. The proposed sliding window model slides forward in hops of batches, while only a single batch is available for processing. The mining algorithm processes events upon their arrival and discovers the top frequent episodes over a window consisting of several batches in the immediate past. A discovered pattern is characterized both by its frequency and by the tendency to persist over time. Experimental evaluation on real and synthetic datasets shows the advantage of the proposed approach over the baselines, in terms of accuracy, performance and memory-consumption.

Another temporal mining approach is proposed in [Sakurai et al. 2008], which focuses on discovering event patterns in textual datasets with time information. Particularly, seven types of time constrains are introduced (e.g., time constraint between two specific events or between an event and the previous or the next one), that can be applied on various types of combinations of events, even if the events are not neighboring. These constraints help analysts filter the discovered patterns and finally get only the interesting ones providing new knowledge. Evaluation results reveal the validity and completeness of time constrains, though the authors mention that they may not be optimal for all datasets.

3.3.3 Logic-based Approaches

Logic-based approaches exploit inferencing to discover patterns defining respective association rules. In [Sim et al. 2010] a pattern discovery approach is proposed mapping logical equivalences based on propositional logic. In particular, a rule mining framework is introduced, generating coherent application domain independent rules for a given dataset that do not require setting an arbitrary *minsup*. The proposed coherent rules mining framework makes use of contingency tables (Table 3.2) displaying frequency distributions of candidate patterns *X* and their negations $\neg X$ as antecedents and the caused event *Y*, as well as its negation $\neg Y$ as consequences. These frequencies (i.e., *a, b, c* and *d*) are further exploited to infer coherent rules among the considered patterns and the caused event, by investigating corresponding *pseudoimplication of equivalence (pe)*, i.e., (i) $X \Rightarrow Y$, (ii) $X \Rightarrow \neg Y$, (iii) $\neg X \Rightarrow Y$; and (iv) $\neg X \Rightarrow \neg Y$. The first and the last pseudoimplications both invigorate the considered coherent rule correlating pattern *X* with caused event *Y*, as the absence of both of them also empowers their interrelationship. The evaluation results reveal that while the proposed framework is more expensive compared to the *Apriori* algorithm because it considers both the positive and negative association rules. Its accuracy results are significantly more accurate and the *minsup* and confidence level of *Apriori* should be very carefully defined to provide all the enclosed patterns, even with a much more exhaustive memory consumption.

Another logic-based approach is proposed in [Artikis et al. 2010; 2012] proposes an event calculus (EC) dialect for efficient run-time recognition that is scalable to large data streams and exploits main EC predicates to discover specific activities. In particular, the authors propose an efficient dialect on the Event Calculus (*EC*), named *RTEC* for run-time reasoning, in order to identify composite events and recognize activities. The proposed dialect employs a number of novel

Table 3.2: A sample contingency table for pattern X and critical event Y

Frequency	Y	$\neg Y$
X	a	b
$\neg X$	c	d

implementation techniques enabling efficient CE recognition, scalable to large streams of simple time-stamped simple derived events. The evaluation results reveal a significant performance in supporting real-time reasoning in most of today's applications. Moreover, in [Artikis et al. 2013] the authors address the issue of uncertainty in transportation systems, using the RTEC reasoning tool to identify regions of uncertainty within a stream of multiple sources and generate common composite events. The approach is evaluated on real data stream comprising traffic events from the city of Dublin, showing efficient self-adaptation capabilities in discarding the corresponding sensors that cause uncertainty during the CE recognition process.

3.3.4 Comparison

Most of the algorithms mining frequent itemsets in large datasets make use of a minimup support (*minsup*) threshold, i.e. a pattern frequency should exceed a low minimum support in order to formulate an association rule. Therefore, these approaches (i.e., mathematical, statistical and temporal approaches as defined below) suffer from many issues [Artikis et al. 2012, Sim et al. 2010]: (i) While every pattern discovery dataset holds an ideal *minsup* it is not an easy task to find it [Webb and Zhang 2005]. (ii) Fluctuations in the *minsup* threshold may result in different patterns, even for the same dataset. Consequently, the accuracy in finding association rules is subject to the accuracy in determining the *minsup*. (iii) Some rare events of the dataset may be actionable, i.e. although a series of specific events does not occur frequently, their occurrence is related to a violation of a specific metric. Thus, if the user defines a high *minsup* this pattern will not be discovered, while a low *minsup* may result in many meaningless patterns. (iv) Even if an ideal threshold is determined, the discovered association rules cannot be further ranked without the use of a confidence value.

Logic-based approaches which do not suffer from the above limitations, posed by the identification of the optimum *minisup*, have some additional advantages: (i) they enable efficient ranking of the discovered patterns, (ii) they can discover potential patterns that should be considered for future executions; and finally (iii) machine learning techniques can also directly be employed to discover patterns. Consequently, taking into account the above limitations of the classic pattern discovery approaches defining a *minsup* threshold, we argue for the use of a logic-based approach for discovering event patterns in the execution history of a SBA.

3.4 Technical Fundamentals

This section provides the fundamental technical information, so that the interested reader harmonically follows the description and the implementation details of the proposed framework. Very briefly Table 3.3 presents the exploited specific languages (DSLs) and techniques, analyzed in detail in the next subsections, and their usage in the current work.

WSLA

This section provides a brief overview over specific parts of the WSLA specification language [Keller and Ludwig 2003], used to specify SLAs in a flexible and individualized way, after providing a brief overview of the most widely used SLA languages.

There is a number of available specification languages used to define SLA doc-

Table 3.3: The exploited languages and techniques and their usage

DSL / technique	**Usage**
Web Service Level Agreement (WSLA)	Describes the SLA documents with the incorporated SLOs
OWL-Q	Describes the monitored properties / metrics of the SBA
Drools Rule Language (DRL)	Used to realize the association rules between the causing event patterns and the caused SLO violation, as well as the adaptation rules
Esper Event Processing Language (EPL)	Used to express the detected event patterns
KairosDB Time-series Database (TSDB)	Stores the timely-synchronized monitoring events and extracts aggregate values
Logic-based Pattern Discovery (LBPD)	Used to discover detrimental event patterns causing SLO violations

uments. Here we make a comparison of the available approaches, used for specifying and managing SLAs, in order to argue for the use of WSLA. The assessment criteria used are the following:

- **Requirement specification:** Both Web service clients and providers need to specify non-functional requirements and offers. The specification should ensure that the compatibility and comparability of the specifications are done by clients and service providers.
- **Class of service:** QoS parameters differ in quality, quantity and the corresponding monetary charge. Grouping similar parameters into a class or category that characterize a service will ease the utilization of the service.
- **QoS aspects:** A Web services related framework should support more than

	Reqs specification	Class of service	QoS aspect	Flexibility
WSLA[Keller and Ludwig 2003]	++	✓	+	++
WS-Agreement [Andrieux et al. 2007]	++	✓	+	++
WSOL [Tosic et al. 2002]	++	++	+	+
SLang[13]	+	✗	+	+
UX [Zhou et al. 2003]	✓	✗	✓	✓
UDDIe [ShaikhAli et al. 2003]	✓	✗	✓	✓

Table 3.4: Comparison of SLA specification languages / approaches

the classical QoS parameters such as jitter and bandwidth. Aspects such as security, reliability, transaction as well as custom defined aspects should also be considered.

- **Flexibility:** An approach should be easy to use, extensible and standards conforming.

Table 3.4 compares the SLA specification languages according to these criteria. The ✓ mark indicates a satisfaction of the requirement, ✗ marks poor or no available aspect. The '+' mark determines the extent to which the specific capability is supported. One '+' means good concept, while two '+'s define excellent concept.

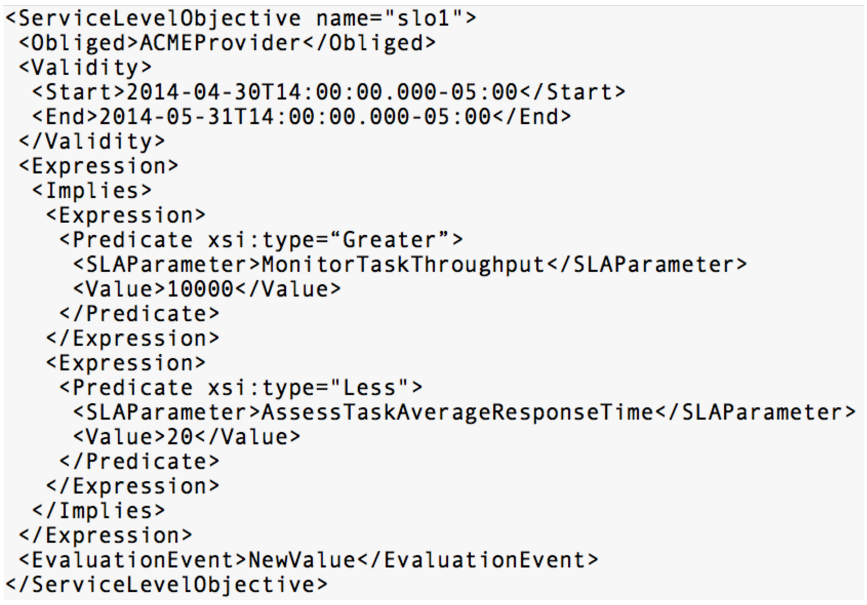
Considering the contents of the above comparison, we can conclude that WSLA and WS-Agreement are the best solutions as they meet almost all criteria. Slang is a moderate solution as it has a good concept of most criteria and UX and UDDIe are the least used approaches. We decided to use WSLA, among the first two approaches, because it supports the definition of Service Level Objectives that are appropriate in our work and it has also been exploited by our previous work presented in section [Zeginis 2009]. In contrast to WSOL that supports only the creation of service offerings and definition of QoS constraints, the WSLA explic-

[13]http://uclslang.sourceforge.net/

itly defined all the metrics with the agreed values. Finally, WSLA is an XML-based language that is very easy in use and is supported by many open-source tools.

Figure 3.4 defines a sample SLO for the traffic management running example, in the WSLA language. Typically, a SLO has the following elements:

- The *Obliged* is the name of a party that is in charge of delivering what is promised in this guarantee.
- One or many *ValidityPeriods* define the period in which the guarantee is applicable.
- A *logicExpression* defines the actual content of the guarantee, i.e., what is asserted by the service provider to the service customer. A logic expression follows first-order logic. Expressions contain the usual operators and, or, not, etc., which connect predicates or, again, expressions. Predicates can have SLA parameters and scalar values as parameters. By extending an abstract predicate type, new domain-specific predicates can be introduced as needed. Similarly, expressions could be extended to contain variables and quantifiers. This provides the parties with the expressiveness to define complex states of the service.
- A SLO may have an *EvaluationEvent*, which defines when the expression of the service level objective should be evaluated. The most common evaluation event is NewValue, each time a new value, for a SLA parameter used in a predicate, is available.
- Alternatively, the expression may be evaluated according to a *Schedule*, which is a sequence of regularly occurring events. It can be defined within a guarantee or a commonly used schedule.



```
<ServiceLevelObjective name="slo1">
 <Obliged>ACMEProvider</Obliged>
 <Validity>
  <Start>2014-04-30T14:00:00.000-05:00</Start>
  <End>2014-05-31T14:00:00.000-05:00</End>
 </Validity>
 <Expression>
  <Implies>
   <Expression>
    <Predicate xsi:type=“Greater”>
     <SLAParameter>MonitorTaskThroughput</SLAParameter>
     <Value>10000</Value>
    </Predicate>
   </Expression>
   <Expression>
    <Predicate xsi:type="Less">
     <SLAParameter>AssessTaskAverageResponseTime</SLAParameter>
     <Value>20</Value>
    </Predicate>
   </Expression>
  </Implies>
 </Expression>
 <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Figure 3.4: A sample WSLA document of the Traffic Management application

OWL-Q

As far as the definition of the monitored properties/metrics of the Web services is concerned, we exploit *OWL-Q* [Kritikos and Plexousakis 2006], which is a semantic, rich and extensible QoS description model for Web services, that is based on Web Ontology Language (OWL) [OWL 2012]. It is designed modularly, incorporating several independent facets, each one focusing on a particular aspect of QoS-based Web service description. There are facets regarding the connection of OWL-Q with OWL-S, the description of QoS offers and requests, the QoS metric model and the definition of constraints. The exploited structure of OWL-Q is presented in Figure 3.5.

A set of OWL-Q facets are exploited: The QoSSpec class (belongs to the Basic Facet) represents the actual QoS description of a Web service. It describes the security and transaction protocols used, the cost of using the service and the associated currency for the cost, the validity period of the offer or demand and an arbitrary OpenMath expression. This expression represents what is or must be guaranteed and contains variables, which are associated to QoS Metrics.

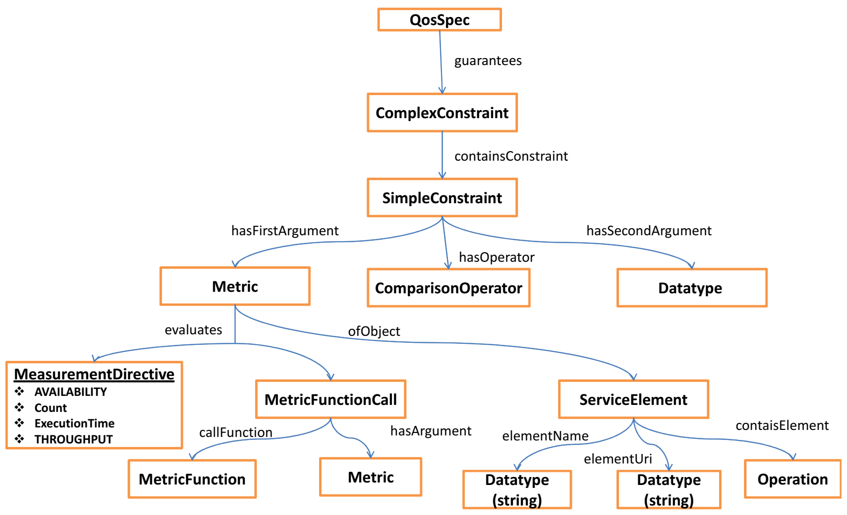


Figure 3.5: Exploited OWL-Q structure

The QoS Metric Facet describes all the appropriate classes and properties used for a proper formal definition of a QoS metric. This metric facet is actually an upper ontology representing any abstract QoS metric. A specific QoS metric can be created by refining the *QoSMetric* class. The QoSMetric is one of the most important classes of OWL-Q representing a QoS metric. The values of a QoS metric are provided by a service provider, a requester or a third-party. These values measure a *QoSProperty* on a specific *ServiceElement*. The value type of a QoSMetric is an instance of the *QoSValueType* class while the unit of the value is an instance of the Unit class. It can be a simple QoS metric measured by a *MeasurementDirective* or a complex one. *ComplexMetrics* derive from other metrics with the help of a *OMFunction*. The Facet Function describes all the appropriate concepts and properties for the proper definition of metric functions. The OMFunction class is the basic concept that represents a QoS Metric Function. The Measurement Directive Facet describes the concept of a measurement directive, which is used for the measurement of simple metrics. A *MeasurementDirective* is composed of a URI that describes where and how to get a value of a resource property.

Drools Rule Language

Drools Expert [Drools 2014] is a declarative, rule-based, coding environment that uses Drools Rule Language (DRL) to express rules in both natural language as well as in XML-format. This language is exploited to formally express the adaptation/scalability rules of our approach, as well as the association rules, interrelating monitoring events. The structure of a drl file in natural language is simple and may contain multiple rules. Its syntax is straightforward and easy to learn. The listing below defines a scalability rule of the traffic management running example in DRL:

Listing 3.1: Sample scalability rule in DRL

```
rule "scale_out-response_time_availability"
    when
        Average_respone_time(MonitorService) > 3sec || Availability(DeviceConfigService) < 99,9%
    then
        scale_out(2 VMs) //if available by the cloud provider
end
```

When new measurements (i.e. facts) are inserted in the Drools working memory, the inference engine checks which rules should be fired based on their matching with the condition part of the rule. Figure 3.6 shows the production rule system of the Drools Rule Engine.

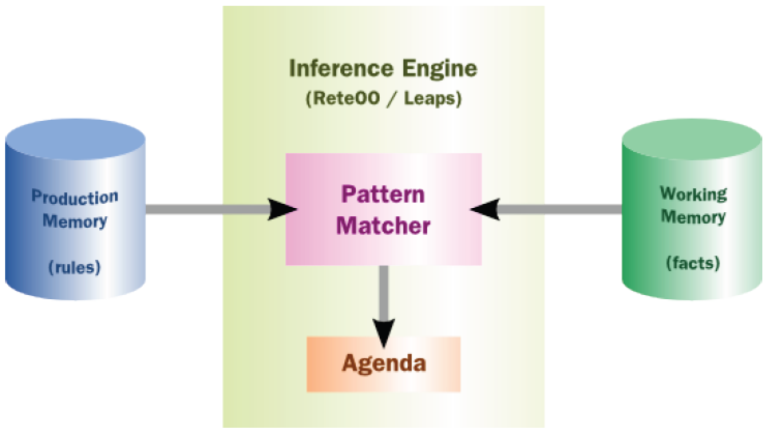


Figure 3.6: The production rule system of Drools Rule Engine

Esper Event Processing Language (EPL)

Espertech Esper[14] is a stream-oriented Complex Event Processing (CEP) engine that provides the *Event Processing Language (EPL)*, much alike SQL, allowing expressing rich event conditions and correlations. The EPL enables expressing complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, sorting and pattern detection. In the proposed framework it is used both for the assessment of the aggregate metrics as well as for the event pattern detection, enabling proactive adaptation. Listing 3.2 defines two sample assessment EPL queries from the traffic management running example, as well as a sample pattern definition in Esper. This pattern comprises two events, discerned by their *eventID*, leading to a specific SLO violation.

Listing 3.2: EPL statements and pattern definition in Esper

```
select avg(response_time) as MSavgexectime from MonitorTaskExecTime where MSavgexectime > 2sec

select availability as ASavailability from AssessServiceAvailability where availability < 99,9%

String patternText = "every a=EVENT_MEASUREMENT(eventID=33453) -> b=EVENT_MEASUREMENT(eventID
    =32451)"
```

Time-series Database (TSDB)

Storing time-based events in relational databases is problematic, as table cardinality and disk space can prohibitively increase with new event arrivals, and indices may become so large that they cannot fit in caches, making data retrieval significantly slow [Deri et al. 2012]. Standard solutions include stream processing engines [Bo 2011] and time-series databases (TSDBs) [Keogh et al. 1993]. The former aim to meet stringent latency requirements when performing continuous queries on the streaming data and to minimize processing cost for large data sets. TSDBs differ in that they focus more on persistent storage of events and in performing *roll-ups* (e.g., aggregated metrics such as average, max, min) for user-

[14]http://www.espertech.com/products/index.php

	open source	data points visualization	roll-ups	real-time monitoring	Cloud support	Scalability	Storage efficiency
KairosDB[15]	✓	✓	✓	✓	✓	+++	+++
openTSDB[16]	✓	✓	✓	✓	∼	+++	++
TempoDB[17]	✗	✓	✓	✓	✓	+++	++
tsdb[18]	✓	✗	✗	✓	✗	+++	++
Cube[19]	✓	✓	✓	✗	∼	++	+
kdb+[20]	✗	✓	✓	✓	∼	+++	++

Table 3.5: Comparison of TSDBs

specified intervals. Other capabilities of TSDBs include the proper handling of timezones, the temporal order of data queries, time-ranged data access and efficient storage. Our architecture aligns with the latter and thus uses TSDBs.

A variety of commercial and open source TSDBs can be used to handle timestamped events. Table 3.5 compares prominent TSDBs on the following features: (i) open-source availability, (ii) data points visualization, (iii) roll-ups / aggregation capabilities, (iv) real-time monitoring (v) Cloud support (vi) scalability; and (vii) storage efficiency. The ✓ mark indicates a satisfaction of the requirement, ✗ mark indicates lack of the requirement while ∼ shows uncertainty. The '+' mark determines the extent to which the specific capability is supported. One '+' means low support, while three '+'s defines full support.

Among the open-source TSDBs, KairosDB stands out for providing all features needed by our monitoring and adaptation framework for the monitoring events storage. Cube and tsdb are two free and open source solutions but the former one does not provide run-time monitoring and sufficiently efficient storage, while the

[15]https://code.google.com/p/kairosdb/
[16]http://opentsdb.net/
[17]https://tempo-db.com/
[18]https://code.google.com/p/tsdb/
[19]https://square.github.io/cube/
[20]http://kx.com/

main limitation of the latter one is the lack of aggregation and visualization tools. kdb+[21] is a very powerful time-series database focusing on financial institutions to analyze and monitor both real-time and historical data, but it is a commercial product. We selected KairosDB for the proposed MA framework, as it provides distributed storage features for the time-stamped monitoring events and it is the most scalable solution, simultaneously offering highly efficient storage via the Cassandra[22] database. Although it is not directly offered as a PaaS add-on for any Cloud platforms, it can easily be deployed on a Cloud VM to collect and handle monitoring events, as well as their roll-ups (i.e., average, min, max over events within specific time periods).

KairosDB outmatches OpenTSDB, that is one of the most widely used TSDBs, as far as the underlying datastore is concerned, i.e., Cassandra and HBase[23], respectively. In KairosDB, string data (i.e., metric names and tags) are written to row keys and the appropriate indices. As Cassandra has much wider rows, there are far fewer keys written to the database. To this end, using the default row size in OpenTSDB's HBase we are able to store 1 hour of monitoring data, while in Cassandra we are able to store 3 weeks data in a single row. In addition, OpenTSDB's main concern is to enable the production of metric graphs. Data is manipulated to make good-appearing and meaningful graphs. For example, interpolation is used to fill in holes left by missing data points and it will grab data points outside the time range to produce again a more meaningful graph.

Logic-based Pattern Discovery

In Section 3.3.3 we have analyzed the logic based approaches for pattern discovery. Among them we have exploited and extended a logic-based approach (we argue for this selection in Section 3.3) presented in [Sim et al. 2010], in order to enable proactive adaptation in the proposed monitoring and adaptation frame-

[21]http://kx.com/kdb-plus.php
[22]`http://cassandra.apache.org/`
[23]http://hbase.apache.org/

work. This work introduces a framework to discover domain knowledge report as *coherent rules*, that associates events with each other.

Assuming that we have an event pattern *X* and a SLO violation *Y*, the association rules that express the causal relationship between *X* and *Y* are: (i) $(X \Rightarrow Y)$ and (ii) $(\neg X \Rightarrow \neg Y)$. The first proves the coherent rule under consideration, while the second also determines it, as the concurrent absence of a pattern and the considered violation event (e.g., aggregate metric event) also interrelates the association rule's root and cause. In order to decide on the coherent rules, this technique exploits contingency tables (see Table 3.2) displaying frequency distributions of candidate patterns and their negations as antecedents, while the specified metric event, as well as its negation as consequences. These tables entries are utilized to determine the association rules, i.e. the sum of frequencies proving the considered coherent rule (i.e. *a* and *d*) should be greater than the sum of frequencies weakening it (i.e. *b* and *c*).

3.5 Conclusions

To sum up, this chapter reviews the related work on Web service monitoring and adaptation, pattern discovery and Cloud modeling, as well as it provides the technical fundamentals, that will ease the reader for the rest of this thesis. In particular, in the first part of the chapter each one of the reviewed topics is analyzed and compared against the requirements set in Section 1.4, while the second part provides the technical information referred to in the remaining chapters.

Chapter 4

ECMAF Framework

Contents

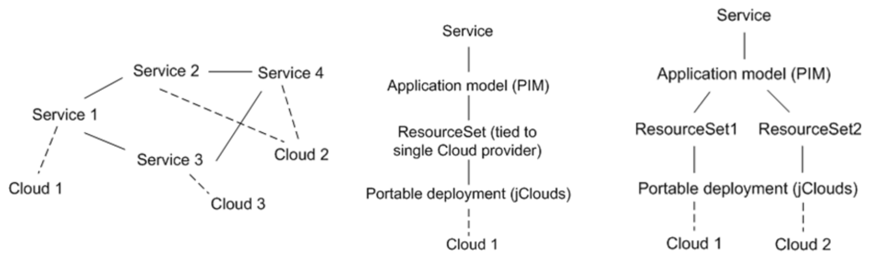
After reviewing related work in cross-layer and multi-Cloud Web service monitoring and adaptation and identifying a set of requirements that must be satisfied, so that these two processes to be successful, accurate and complete, this chapter introduces and analyzes the proposed **E**vent-based **C**ross-layer **M**onitoring and **A**daptation **F**ramework (*ECMAF*) for SBAs. The proposed framework will be used throughout the rest of the dissertation to concretize and place the application of the individual contributions on it. This chapter is organized as follows. Section 4.1 is dedicated to the scope of the proposed framework, while Section 4.2 to the presentation of ECMAF framework and, in particular, to the functionality of each individual component and their interrelationships. Section 4.3 highlights the benefits of the proposed framework and finally, Section 4.4 concludes this chapter and paves the way for the next chapter analyzing the meta-models utilized by the framework.

4.1 Solution's Scope

Before introducing the proposed cross-layer and multi-Cloud monitoring and adaptation framework, this section describes the scope and the prerequisites of our solution. As it is a multi-featured framework combining many technologies and techniques, the user must be aware of them in order to use it properly. An example from the traffic management application is used to exemplify the framework's scope and application domain.

As already discussed in Chapter 1 application developers all the more adopt Cloud infrastructures as the dominant services delivery platform. However, complex SBAs, comprising many simple Web services with different requirements may be deployed on multiple Clouds, thus leveraging the various offerings of the available Cloud providers. In [Baryannis et al. 2013a] we focus on the need to break the current lock-in, experienced by application developers on the Cloud provider they design for and deploy on, and to allow them to simultaneously use several Cloud providers.

Application development for Cloud platforms today follows two main (often complementary) approaches: (i) Composition and use of SaaS instances exported by providers such as Salesforce (CRM and ERM applications), Google (Google Apps), etc.; and (ii) development of the application over middleware offered via PaaS providers (such as Amazon Elastic Beans) or at a lower level of abstraction, over Infrastructure-as-a-Service (IaaS) providers (such as Amazon EC2 or Microsoft Azure). In the former approach (Figure 4.1(a)), each SaaS instance can be implemented and deployed over a different Cloud provider, naturally supporting heterogeneity (although at a fairly coarse level of granularity). In the latter approach (Figure 4.1(b)), a SaaS instance is typically developed using model-driven software engineering methodologies targeting individual Cloud providers. A problem with current methodologies is that –although they address portability via the use of



(a) SaaS composition (b) Single-Cloud deployment (c) Multi-Cloud deployment

Figure 4.1: Cloud application development approaches

generic platform APIs such as jclouds[1] – they tie all Cloud resources (referred to as *ResourceSets*) to a single Cloud provider and thus preclude deployment of the application on multiple Cloud providers.

There are several reasons justifying deployment of complex applications on multiple Clouds. For instance, an application may have dependencies on software components or services offered by different Cloud providers. In addition, different components of an application may have various resource requirements that are best satisfied by different Cloud providers. For instance, provider A may offer specialized VMs of a certain kind –e.g., featuring graphics accelerators, solid-state storage devices, etc.– while provider B may specialize in another –e.g., higher core-count or dynamically reconfigurable VMs–. Cloud providers may also differentiate on their offered cost for different types of resources (e.g., CPU is cheaper on provider A while provider B delivers cheaper I/O throughput). Two components of the same application may need to be deployed to different geographical zones for proximity reasons, to stay local to these geographies –data, sensors, etc.– or to minimize chances of catastrophic failure. In the latter case, an application provider will typically deploy redundant parts of the application to different Cloud providers. Generally, decomposition of a complex application on multiple Clouds may be either *functional* (different parts of the application logic, placed on

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

different providers) or *data-driven* (redundant functionality with state/data split to different providers) or a combination of them.

The proposed framework applies on both SaaS compositions and single Cloud deployments (subsection 4.2.1) but its main focus is on such multi-cloud setups (subsection 4.2.2), where various VMs are involved, either from the same or from different Cloud providers. Thus, the application developer may choose to deploy the components of her/his SBA on various VMs, according to the individual requirements. In this way, the deployment cost is optimized, as the user does not reserve redundant computational power, that would be used for a single-Cloud deployment satisfying the requirements for all SBA components.

We consider for example a multi-cloud setup of the traffic management running example, analyzed in Section 2.3. This SBA is characterized by strong interdependencies between the layers and raises various events during its lifecycle [Zeginis et al. 2012b]. The main goal of the SBA is to regulate traffic aiming to optimize for particular environmental conditions (e.g., CO_2 levels, temperature and air pollution) drawn from real-time sensor measurements and to properly address car accidents or other incidents impeding normal car flow. Each of these cases is handled by a different sub-process. Some of the system roles are the traffic management authority, the drivers, the pedestrians and the rescue forces (traffic police, fire brigade).

Other events, other than the ones introduced in Section 2.3, could come from internal or third party services and their composition (e.g., calendar service, SMS service, incident assessment service) and can be either functional or non-functional (QoS events). Furthermore, events regarding the whole business process, as well as violations of KPIs may arise. Thus, based on the aforementioned example, it is clear that the framework's scope is on composite SBAs deployed on multiple Clouds. These multi-Cloud setups may be based on public, private or hybrid Clouds, providing the minimum requirements for the individual SBA components.

As far as the monitoring and adaptation processes are concerned, it is crucial that (i) the deployment VMs provide the required monitoring mechanisms to produce events (IaaS layer), necessary to drive the adaptation and evolution path of the Multi-Cloud lifecycle, (ii) the VMs are properly time-synchronized; and (iii) the application developer provides simple adaptation/scalability rules for the most common monitoring events emitted by her/his application.

4.2 ECMAF Architecture

After having analyzed the scope of the proposed solution, this section introduces the **E**vent-based Cross-layer Monitoring and Adaptation Framework (ECMAF) for SBAs. First, a simplified version of the framework focusing on SaaS compositions is presented in subsection 4.2.1 [Zeginis et al. 2012b] and in subsection 4.2.2 we focus on a more extensive version applicable for Multi-Cloud deployments [Zeginis et al. 2013a; 2014a]. Both versions' functionality are exemplified by the traffic management running example.

4.2.1 Single-Cloud Deployment

Figure 4.2 presents the cross-layer architecture of the proposed ECMAF framework for SaaS compositions deployed either on private infrastructure or on a single Cloud (private or public). This framework comprises a Monitoring Engine able to collect the monitoring events during the service execution, an Adaptation Engine able to perform adaptation actions and an Execution Engine. The first two engines communicate with each other via events through a publish/subscribe mechanism.

Monitoring Engine. The Monitoring Engine comprises a **Monitor Manager** and a number of individual **Monitoring Components**. Each of the latter components is assigned to detect events at a specific SBA layer and to immediately deliver them to the Monitor Manager. The Monitor Manager, in turn, continuously delivers information about the service execution produced by the Execution Engine while collecting events from the Monitoring Components. The Monitor Man-

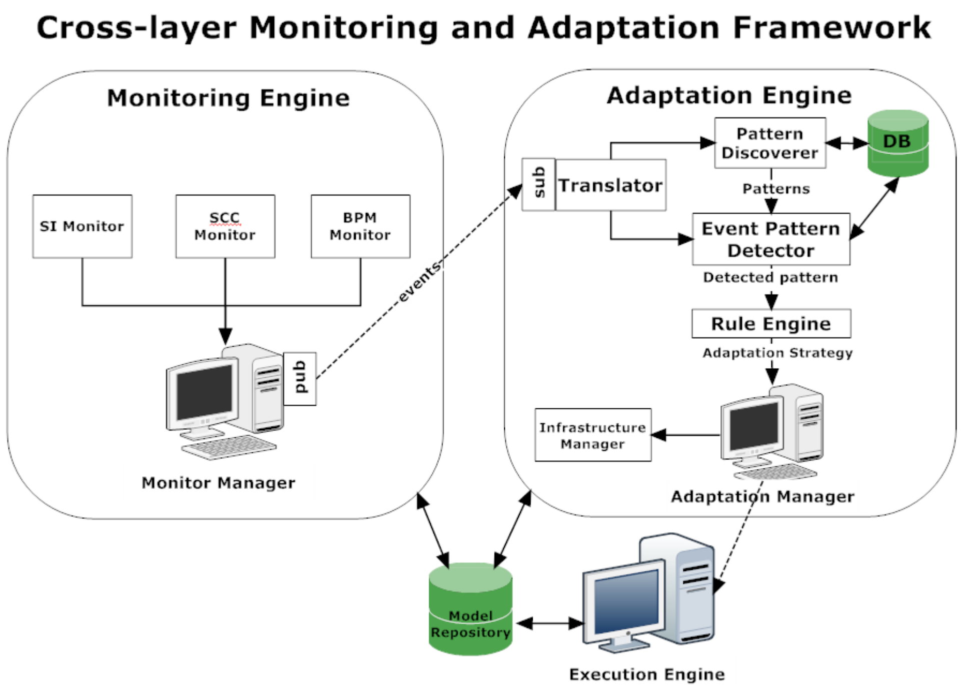


Figure 4.2: Architecture of ECMAF for SaaS compositions or single-Cloud deployments

ager communicates with the Translator component (see description below) via a publish/subscribe mechanism. A required monitoring event is delivered to the Translator as soon as it is detected. It is imperative to send the events in the order that they are received so as to have an as reliable as possible pattern discovery mechanism. As there are many detected monitoring events, specific techniques are required to ensure this, such as event timing [Mok and Liu 1997] and clock synchronization [Kopetz and Ochsenreiter 1987].

Adaptation Engine. The Adaptation Engine comprises a number of components supported by suitable repositories:

- The **Translator** receives the events sent by the Monitor Manager and translates them into a suitable for the Event Pattern Detector and Pattern Discoverer components format, conforming to the XML or JSON schemas[2], stored in the model repository. In addition, it incorporates a *subscription mecha-*

[2]available at www.ics.forth.gr/ zegchris/schemas

nism, enabling the adaptation engine to subscribe to the required events and collect them through a predefined socket component, managing the communication between the publisher and the subscriber. For instance, an adaptation engine tied to a municipal Cloud in our traffic management example is not interested in events from applications running on other municipal Clouds, thus it subscribes to events related to its own Cloud. Otherwise, gathering other "foreign" events, could lead to useless rules and inaccurate event patterns, discovered by the adaptation engine.

- It is very common that failures at the SI layer lead to other failures and violations both at the same and higher layers, forming a chain of monitoring events. The **Pattern Discoverer** is capable of processing the monitored events stream, after the translation process and discovering event patterns causing specific SLO violations. These patterns are stored into the *Patterns Repository* with extra information regarding the discovery time, the execution instances considered to extract the specific pattern and the sub-patterns that can similarly lead to the same violation. They can also be further exploited by the Event Pattern Detector and again by the Pattern Discoverer to enhance its performance by eliminating the already discovered patterns.

- It is desirable to detect those event patterns causing service failures by exploiting specific mechanisms (e.g. pattern matching ones [Karp and Rabin 1987]). These patterns are detected by the **Event Pattern Detector**, which, in turn, detects at run-time the patterns extracted by the Pattern Discoverer and passes them immediately to the Working Memory of the Rule Engine. In case a pattern is not detected at one execution but a sub-pattern is detected in every SBA execution, then the Rule Engine is informed about it, so as to take preventive actions, before the actual violation occurs. Going back to the traffic management running example, if the discovered pattern $\{e_1, e_2, e_4 \rightarrow E3\}$ (see Section 2.3 – E_3 refers to the violation of the corresponding aggregated metric) is never detected, but its sub-pattern $\{e_1, e_2 \rightarrow E3\}$ appears frequently, then whenever the sequence of events $\{e_1, e_2\}$ appears in the monitoring events stream, suitable adaptation actions should be triggered.

- The **Rule Engine** is responsible for mapping the detected patterns to suitable adaptation strategies. Our approach relies on using a mapping technique, exploiting simple manual mapping from single monitoring events to specific adaptation strategies. Thus, the dynamically detected patterns trigger an adaptation rule dictating the enactment of a series of corrective actions, preventing the actual SLO violation from happening. As such, it exhibits *proactive SBA adaptation* capabilities.

- The **Adaptation Manager** executes the adaptation strategy exported by the Rule Engine with the aid of two components: (i) the **Infrastructure Manager** is able to treat malfunctions regarding the SI layer, which is the main source of many service failures; and (ii) the **Model Repository** supplies the appropriate information, such as service descriptions and requirements, SBA components and their dependencies, triggerable adaptation actions per component, metric and SLA models, so as to fulfill the supported adaptation strategies. This is a live repository where models are modified based on modifications of the system at different layers or modifications according to the user requirements. The Execution Engine supports the adaptation process, especially for strategies regarding the BPM and SCC layers.

Running Example

This case study is inspired by the traffic management running example analyzed in Chapter 2 and describes a traffic management system designed to manage normal traffic situations as well as emergency cases. Such emergency case handling includes several different actions, such as directing the rescue forces to the accident location and managing traffic deviations. Figure 4.3 and Figure 4.4

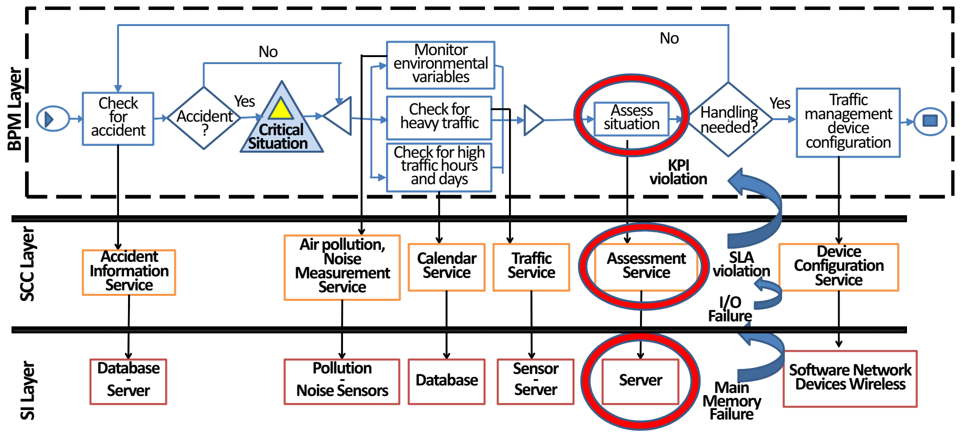


Figure 4.3: Layers' interaction during normal traffic conditions

respectively illustrate these two cases. Each figure depicts the three functional layers. In both cases, workflow tasks are executed either manually or by mapping them to Web services. Each service is then mapped to the appropriate infrastructure.

Figure 4.3 illustrates normal traffic conditions, where the system tries to optimize some parameters, such as total noise, overall throughput and air pollution. In particular, the system shall consider different needs, such as the ones of pedestrians and motorists and other factors like heavy traffic, public events, school and working hours, holidays or public regulations which may alter traffic demand and needs during conditions that do not involve emergencies. The system interrupts the normal traffic situation process, when an accident happens and jumps to the critical traffic situation process.

Figure 4.4 depicts a critical traffic situation, in which a serious car accident occurs at the monitored area. In particular, the involved citizens inform the traffic manager that must control the overall traffic situation (control traffic devices, inform citizens) and assess the incident so as to inform the appropriate rescue forces about the accident and direct them to the incident location. Moreover, the traffic manager monitors the environmental variables, such as air pollution and noise. Various adaptation actions could be taken by the traffic manager, as well

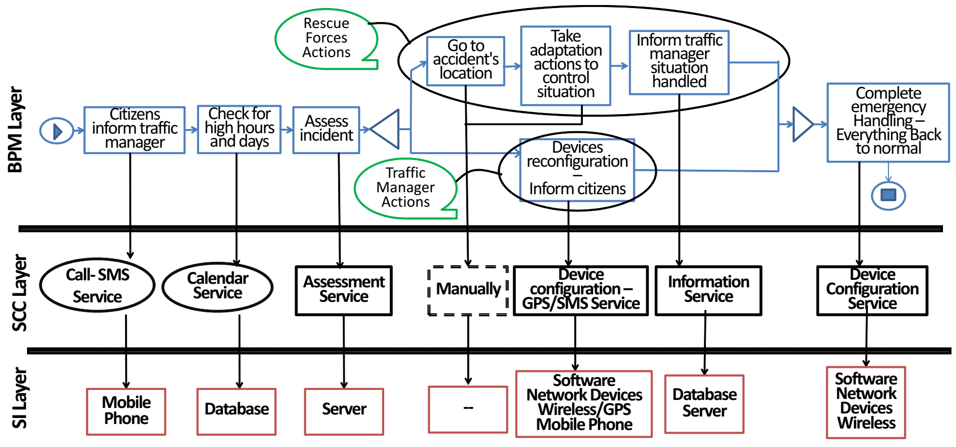

BPM Layer
SCC Layer
SI Layer
Rescue Forces Actions
Citizens inform traffic manager
Check for high hours and days
Assess incident
Go to accident's location
Take adaptation actions to control situation
Inform traffic manager situation handled
Complete emergency Handling – Everything Back to normal
Traffic Manager Actions
Devices reconfiguration – Inform citizens
Call- SMS Service
Calendar Service
Assessment Service
Manually
Device configuration – GPS/SMS Service
Information Service
Device Configuration Service
Mobile Phone
Database
Server
–
Software Network Devices Wireless/GPS Mobile Phone
Database Server
Software Network Devices Wireless

Figure 4.4: Critical traffic situation

as by the rescue forces, such as:

- Traffic devices reconfiguration (e.g., traffic lights) by the traffic manager, in order to reduce stop-and-go traffic. This should also help to keep air pollution low, even if it is not critical during emergency situations.
- Accident reporting to citizens via their devices (e.g, GPS, mobile phones) by the traffic manager to avoid traffic congestion at the accident location.
- Traffic closing/limiting to or from the involved location by the rescue forces.
- Traffic deviation by the rescue forces through alternative places not intended for heavy traffic.

After a complete emergency handling, there is a gradual return back to the normal situation. The rescue forces inform the traffic manager, who updates the system and informs the citizens through their devices.

As already discussed in Section 1.1.1, there are various dependencies among the SBA layers. The occurrence of a failure at one layer may result in a failure at other layers. This work aims at locating the failure event and taking adaptation actions in order to prevent its spread to the other layers, as soon as possible.

Figure 4.3 presents an illustrative example. We suppose that a KPI dictates that the maximum duration of the process should be less than 10 seconds. Furthermore, we suppose that an SLA for the assessment service AS dictating that its maximum execution time must be less than 6 seconds. As shown in the figure, a violation of the respective SLA constraint may cause a violation to the KPI, by considering that the previous process activities do not run longer than 3 seconds. It must also be indicated that AS's execution time is inversely related to the main memory size and the CPU percentage allocated for its execution. Moreover, there is a low limit for the main memory allocated, after which the SLA violation will be unavoidable as the service behavior will be unpredictable and even if it does not fail it will certainly take a longer time to execute. In fact, 2 seconds after the AS's execution, the main memory allocated to it has indeed fallen under the low level of 50 MB.

A monitoring component, running on the server side, where AS is deployed on, detects that the available main memory is not sufficient (SI layer) for AS. At the same time, another monitoring component detects that there is an I/O failure at the SCC layer as AS has produced a wrong output. Both events are first sent to the Translator, which transforms them to the appropriate format and sends them to the Pattern Detector and to the Pattern Discoverer components. Based on the two events received, a specific rule is fired which derives that the best adaptation strategy is to execute another instance of the AS service at a more powerful server and with a better memory and CPU allocation. The suitability of the strategy lies on the fact that by executing a "better" service instance and with better allocation for the hardware resources, the probability that the SLA is not violated becomes very high (as we do not know if another failure may occur in the near future regarding the new instance) and in this way, the KPI violation may also be avoided. Such a rule has been derived by the Event Pattern Detector based on the previous history log. The derived strategy is sent to the Adaptation Manager which executes it with the assistance of the Infrastructure Manager and the Execution Engine.

It is clear from the aforementioned analysis that ECMAF can efficiently handle such a cross-layer scenario. The adaptation actions performed are the appropriate ones, based on the event history and the current context. Moreover, the dependencies among the layers are clearly discerned.

4.2.2 Multi-Cloud Deployment

The ECMAF framework presented until now is applicable on simple service compositions or single cloud deployments and does not take into consideration Cloud deployments, where multiple Clouds are involved (either private, public or hybrid), even from different cloud providers. Figure 4.5 presents an updated version of ECMAF. The main differences between the two versions are: (i) the more comprehensive functionality of the Monitor Manager, (ii) the addition of the time synchronization component, (iii) the incorporation of a time-series database (TSDB) to store the monitoring events; and (iv) the integration of the Rule Manager and the Event Pattern Detector components into a more comprehensive component, named Metrics Aggregator.

As far as the Monitoring Engine is concerned, every VM employs its own monitoring tools, providing monitoring events for the specified metrics at each one of the Cloud and SOA layers. The Monitor Manager that lies on a different VM or on one of the existing VMs, hosting the various components of the SBA, retrieves the monitoring results from the individual monitoring mechanisms, assesses and stores them in a time-series database (TSDB), reporting detected violations via the publish/subscribe mechanism to Adaptation Engine instances. We must mention that each Cloud participates with its own Monitoring Engine and there may be multiple instances of the Adaptation Engine.

One of the main goals of our approach is to identify particular *event patterns* occurring during SBA execution that lead to critical violations so as to enable the selection of the appropriate cross-layer adaptation actions. Since the order of publishing events is significant, monitoring events must be time-synchronized

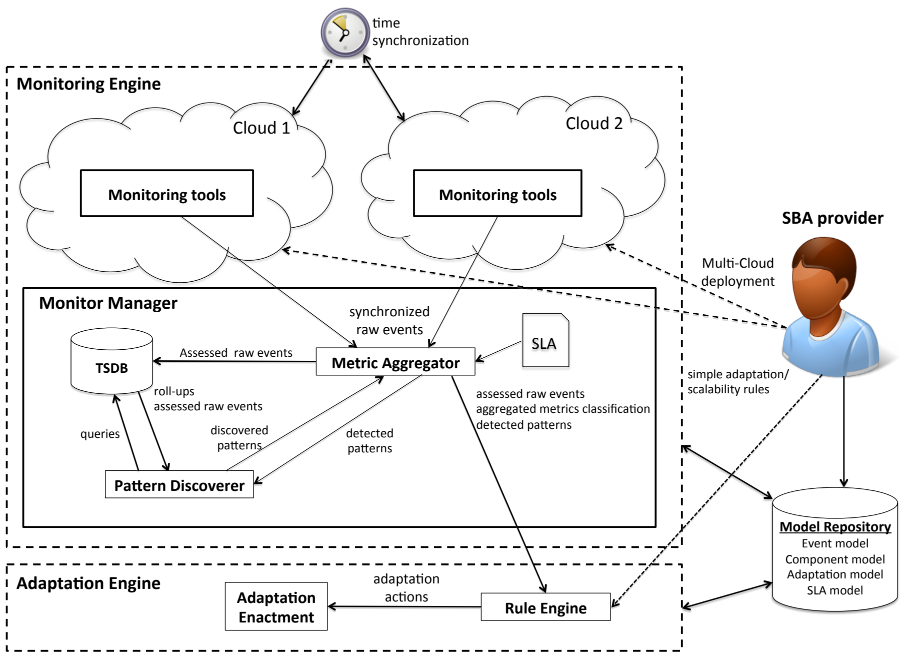


Figure 4.5: ECMAF's multi-Cloud architecture

by the Monitor Manager before being sent to the Adaptation Engine. Time synchronization is particularly important in multi-Cloud settings as standard time synchronization solutions are rarely deployed across Cloud providers. Thus, this updated version of the ECMAF employs an event time synchronization mechanism, responsible of keeping timely synchronized the various machines, either local or virtual, so as to deliver the monitoring events to Monitor Manager with the actual order in which they are detected.

Concerning the storage of the monitoring events, as already analyzed in Chapter 3, we argue for the use of a **Time-series Database (TSDB)**, that offers many useful capabilities for time-ordered data persistence, as well as for providing roll-ups (i.e. aggregate metrics) for specific time intervals. Other capabilities of TSDBs include the proper handling of timezones, the temporal order of data queries, time-ranged data access and efficient storage. Our architecture aligns with the latter one and thus uses (per-Cloud, federated) TSDBs. In this setting, the publish/subscribe mechanism handles the transfer of the raw monitored events and corresponding TSDB roll-ups to the Adaptation Engine. Different instances of the Adaptation Engine may be deployed to distribute the adaptation load across applications or Clouds, with each instance interested only in relevant events and roll-ups.

Finally, the "new" **Metrics Aggregator** component combines and enhances the functionality of the Translator and Event Pattern Detector components. Thus, it is responsible for collecting the synchronized monitoring events, assessing them, as well as of detecting event patterns causing specific SLO violations at run-time and informing the adaptation engine about the detected patterns and the violation of single metrics. It is the central component of the Monitoring engine that provides core functionality for the collection and manipulation of the monitored events.

Running example

In order to exemplify the framework's multi-Cloud functionality, we consider the following scenario, based on the traffic management example. As already discussed in Section 4.1, the optimal deployment of the traffic management applications is on two different Clouds satisfying the individual service requirements; Monitoring and Device Configuration services are deployed on a municipal Cloud with moderate storage capacity and low computational power and the Assessment service on a central Cloud offering high storage and computation levels. Each one of the two Clouds, which are timely synchronized, exhibits a client for the complex event processing (CEP) engine, collecting and assessing the produced events immediately after their occurrence. Assessment is performed by the Metrics Aggregator component, which parses the SLA document, defined in the WSLA language (see Section 3.4) and stored in the Model Repository and then stores the assessed raw events to the TSDB (see Section 3.4), which acts as a repository of the raw measurements/events preserving their ordering. Thus, the monitoring components of the involved Clouds directly interact with the Monitor Manager, which resides on a separate private infrastructure through a CEP server-client mechanism. The pattern discoverer component periodically queries the TSDB and identifies patterns of raw events leading to SLO violations (mapping to specific aggregate metrics). The average metric values, necessary for the pattern discovery process, are also provided by the TSDB, which is capable of providing roll-ups (i.e. aggregate metrics) for specific time intervals. The discovered patterns are stored into a pattern repository, so as the Metric Aggregator can easily retrieve them and detect their occurrence at run-time. Upon pattern detection the Metrics Aggregator urges the Rule Engine of the Adaptation Engine to fire the corresponding adaptation/scalability rule (i.e. proactive adaptation), dictating the application of an adaptation strategy, realized by the Adaptation Enactment component. Raw events are also passed to the Rule Engine by the Metric Aggregator so as to perform reactive adaptation when a SLO violation is detected.

4.3 ECMAF's Benefits

The main benefits of the ECMAF framework presented in the previous sections are the following:

- **Distributed workload**. As there are layer-specific monitoring components that pass monitoring events to the Monitor Manager, monitoring is distributed among the available monitoring mechanisms. In addition, there can be many computer nodes with a separate Monitoring and Infrastructure Manager component that can a handle a portion of the whole monitoring and adaptation workload, as depicted in Figure 4.6.

- **Extensibility**. As services evolve, new monitoring and adaptation techniques are required in order to cope with continuous context changes and other unpredictable malfunctions. This framework can integrate such techniques with the existing ones, while preserving its functionality and integrity.

- **Cross-layer capability**. The framework is able to support all SBA and Cloud functional layers. It incorporates mechanisms to detect events across all

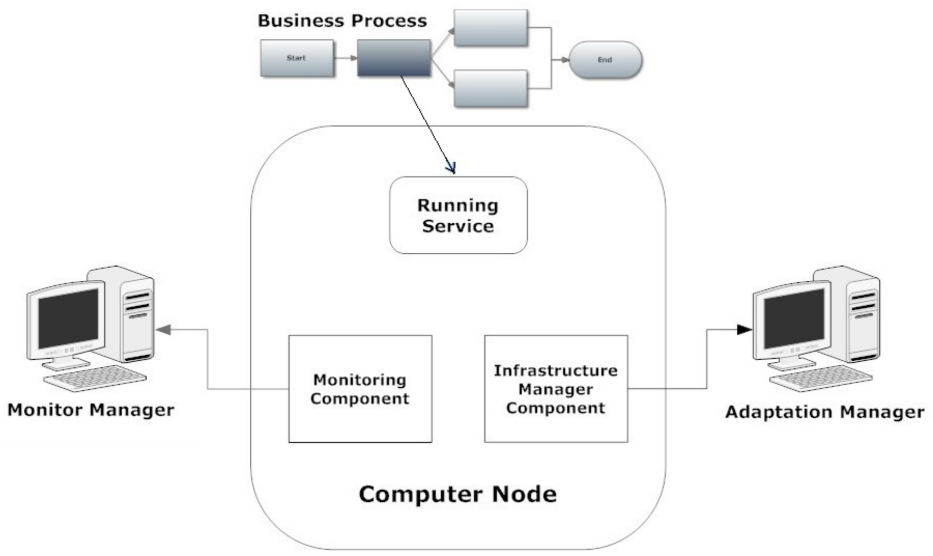


Figure 4.6: Distributed workload performed by a computer node

layers and derive additional events using pattern matching techniques.

- **Multi-Cloud monitoring and adaptation.** Distributed architecture as well as the careful design of the necessary models (see Chapter 5) confers multi-cloud capabilities to the framework.

- **Pro-active adaptation.** The use of pattern matching techniques, as well as the mapping between patterns and adaptation strategies allows for proactively adapting the SBA.

4.4 Conclusions

To sum up, the four main parts of this chapter were dedicated to the analysis of the proposed cross-layer monitoring and adaptation framework (ECMAF). In particular, the first part introduced the scope of ECMAF by explicating the three main deployment types and highlighting the benefits of single and multi-cloud deployments. In addition, a sample deployment of the traffic management applications is utilized to exemplify the solution's scope. The second and third parts present a SaaS composition / single-cloud and a multi-cloud perspective of EC-MAF respectively, providing details about its components' functionality, while elucidating its applicability through the traffic management running example. Finally, the last part pinpoints the main benefits of the proposed framework.

Chapter 5

Meta-Models for Cloud SBA Monitoring and Adaptation

Contents

After presenting the ECMAF framework for cross-layer monitoring and adaptation of multi-Cloud SBAs, this chapter introduces the required meta-models for supporting this framework, stored into the Model Repository. The goal of this chapter is to raise the need and propose new models for the monitoring events, their inter-dependencies, the involved components for a multi-Cloud SBA and the adaptation actions applicable on each of the identified components. All models are presented in Unified Modeling Language (UML). The main benefits of UML that led us to this decision are the following: (i) it keeps as a proper standard in system development process, (ii) it is a formal modeling notation with strongly defined meaning for each element; and (iii) it improves insight and visualization of complex systems (such as a multi-cloud SBA).

The chapter is organized as follows: Section 5.1 presents the event meta-model and a description of its main classes, as well as a taxonomy of the adaptation-related monitoring events. Section 5.2 provides the proposed component meta-model and a description of their classes, while Section 5.3 introduces the adaptation actions meta-model. All these meta-models are accompanied by an instance model (i.e., a UML object diagram) of the traffic management SBA and the mappings to the other instance models.

5.1 Event Meta-Model

In this section we present an event meta-model describing the most common monitoring event types and patterns that occur during the execution of a Cloud SBA. This meta-model (Figure 5.1) is generic enough and extensible to incorporate any other event type defined by domain-specific service providers. Corresponding XML and JSON schemas were designed to guarantee the validity of concrete event models defined in XML or JSON[1].

5.1.1 Meta-Model Description

The main meta-model's class is **Event**, which has an ID attribute, used to uniquely identify a concrete event. Its **CompositeEvent** and **SimpleEvent** subclasses represent simple and composite events, respectively. Composite events consist of two other (simple or composite) events, the *first* and the *second* one, which map to a

[1]www.ics.forth.gr/ zegchris/schemas

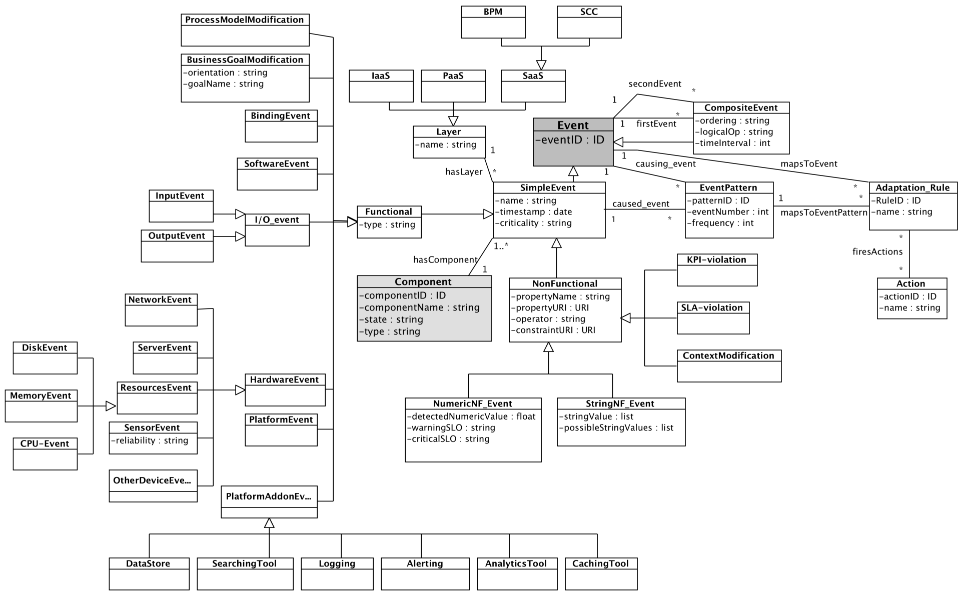


Figure 5.1: The event meta-model

particular *order* (e.g., the first event precedes the second or they have occurred in parallel). As an example of a composite event, consider a hardware event comprising a CPU overload and a low available memory event. Simple events are characterized by their name, timestamp and the ID of the source component and belong to a specific Cloud layer (SaaS, PaaS, IaaS). A SaaS event can be further located at the BPM or SCC layers. Events are also characterized by their importance/criticality as warning, critical or successful events.

A simple event can either be **Functional** or **Non-Functional.** Functional events refer to operational characteristics that define the overall SBA behavior. Non-functional events refer to quality attributes that are either measurable or get distinct qualitative values, such as availability and response time. Two additional different classifications exist for non-functional events: (i) they can be classified as **KPI-violations**, **SLA-violations** or **contextModification** events; and (ii) as numeric or string events. Subclasses that have been defined for functional events include: **Process Model Modification**, **Business Goal Modification**, **Software Event**, **I/O event**, **Hardware event** and **Platform event.** Finally, the **EventPat-**

tern class represents the event patterns discovered during the execution of Cloud SBAs and leading to critical violation events. Each event pattern has a (composite or simple) *causing event* and a simple *caused event*. It is also characterized by a pattern ID, the number of causing events participating in it, as well as its frequency of occurrence.

Finally, this meta-model describes all the involved parts of an Event-Condition-Action (ECA) triplet, totally represented by an **Adaptation_Rule**, which is characterized by its unique ID. An Event corresponds to an EventPattern, a Condition to the SLO, while each Rule fires a specific **Adaptation Rule**, uniquely identified by its ID. An extensive analysis of the meta-model's classes, their properties and their interrelationships can be found in Appendix A.

5.1.2 Traffic Management Event Model

In Section 4.1 we have described a scenario of the traffic management SBA, showing the great amount of metric violations that may be produced in all the functional layers of an SBA. Figure 2.4 in Section 2.3 presents a distribution of the traffic management application's components and the source of the detected events e_1–e_7. In this setup, an example of an event pattern comprises a composite causing event {we_1,ce_2,ce_4,we_3} and a caused average execution time violation of Tasks T_A and T_D, i.e., event E_7. This event pattern can be described through the proposed event meta-model, identifying the order, the layer and the source component of all the involved events within this 4-size pattern.

Figure 5.2 depicts the corresponding instance model of the identified pattern (patternID=2001) and the fired adaptation rule triggering a suitable adaptation strategy, i.e., a scaling action at the Flexiant VM, as well as redo/restart activities of the hosted tasks/services. The structure of this complex adaptation strategy and the ordering of the lower level adaptation actions are described in the proposed adaptation action model in Section 5.3.

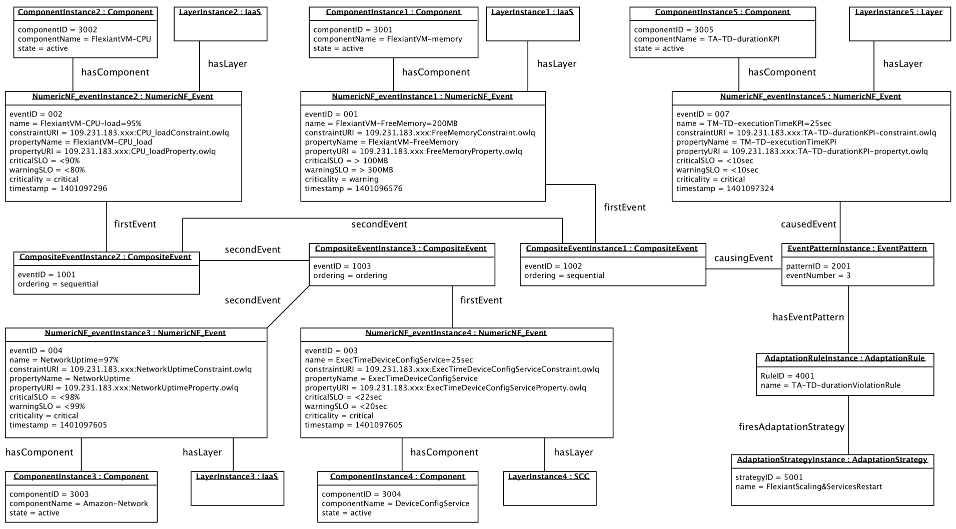

ComponentInstance2 : Component
componentID = 3002
componentName = FlexiantVM-CPU
state = active
LayerInstance2 : IaaS
hasComponent
hasLayer
NumericNF_eventInstance2 : NumericNF_Event
eventID = 002
name = FlexiantVM-CPU-load=95%
constraintURI = 109.231.183.xxx:CPU_loadConstraint.owlq
propertyName = FlexiantVM-CPU_load
propertyURI = 109.231.183.xxx:CPU_loadProperty.owlq
criticalSLO = <90%
warningSLO = <80%
criticality = critical
timestamp = 1401097296
ComponentInstance1 : Component
componentID = 3001
componentName = FlexiantVM-memory
state = active
LayerInstance1 : IaaS
hasComponent
hasLayer
NumericNF_eventInstance1 : NumericNF_Event
eventID = 001
name = FlexiantVM-FreeMemory=200MB
constraintURI = 109.231.183.xxx:FreeMemoryConstraint.owlq
propertyName = FlexiantVM-FreeMemory
propertyURI = 109.231.183.xxx:FreeMemoryProperty.owlq
criticalSLO = > 100MB
warningSLO = > 300MB
criticality = warning
timestamp = 1401096576
ComponentInstance5 : Component
componentID = 3005
componentName = TA-TD-durationKPI
state = active
LayerInstance5 : Layer
hasComponent
hasLayer
NumericNF_eventInstance5 : NumericNF_Event
eventID = 007
name = TM-TD-executionTimeKPI=25sec
constraintURI = 109.231.183.xxx:TA-TD-durationKPI-constraint.owlq
propertyName = TM-TD-executionTimeKPI
propertyURI = 109.231.183.xxx:TA-TD-durationKPI-propertyt.owlq
criticalSLO = <10sec
warningSLO = <10sec
criticality = critical
timestamp = 1401097324
firstEvent
secondEvent
firstEvent
causedEvent
CompositeEventInstance2 : CompositeEvent
eventID = 1001
ordering = sequential
secondEvent
CompositeEventInstance3 : CompositeEvent
eventID = 1003
ordering = ordering
CompositeEventInstance1 : CompositeEvent
eventID = 1002
ordering = sequential
causingEvent
EventPatternInstance : EventPattern
patternID = 2001
eventNumber = 3
secondEvent
firstEvent
hasEventPattern
NumericNF_eventInstance3 : NumericNF_Event
eventID = 004
name = NetworkUptime=97%
constraintURI = 109.231.183.xxx:NetworkUptimeConstraint.owlq
propertyName = NetworkUptime
propertyURI = 109.231.183.xxx:NetworkUptimeProperty.owlq
criticalSLO = <98%
warningSLO = <99%
criticality = critical
timestamp = 1401097605
NumericNF_eventInstance4 : NumericNF_Event
eventID = 003
name = ExecTimeDeviceConfigService=25sec
constraintURI = 109.231.183.xxx:ExecTimeDeviceConfigServiceConstraint.owlq
propertyName = ExecTimeDeviceConfigService
propertyURI = 109.231.183.xxx:ExecTimeDeviceConfigServiceProperty.owlq
criticalSLO = <22sec
warningSLO = <20sec
criticality = critical
timestamp = 1401097605
AdaptationRuleInstance : AdaptationRule
RuleID = 4001
name = TA-TD-durationViolationRule
firesAdaptationStrategy
hasComponent
hasLayer
hasComponent
hasLayer
ComponentInstance3 : Component
componentID = 3003
componentName = Amazon-Network
state = active
LayerInstance3 : IaaS
ComponentInstance4 : Component
componentID = 3004
componentName = DeviceConfigService
state = active
LayerInstance4 : SCC
AdaptationStrategyInstance : AdaptationStrategy
strategyID = 5001
name = FlexiantScaling&ServicesRestart

Figure 5.2: Event model instance of the traffic management running example

5.1.3 Adaptation-Related Monitoring Events

Many of the proposed monitoring approaches can detect different event types. These events deliver information about the SBA evolution and its context change. They are used to indicate whether the SBA execution evolves normally and whether there are some deviations or even violations of the desired or expected functionality. Most events are recurring, usually with the same order, during service executions. Thus, it is desirable to introduce a taxonomy of SBA monitoring events to enable the mapping between these events and the suitable adaptation strategies as well as the event derivation applied in the proposed framework.

The taxonomies of common monitoring events proposed are either generic or domain-specific (e.g. real-time SBAs). [Popescu et al. 2010] introduces an event taxonomy for three possible application layers: (i) the organization layer, (ii) the behavior layer; and (iii) the service layer, to semi-automate the discovery and selection of adaptation templates required to fulfill complex adaptation requirements. [Kongdenfha et al. 2009] categorizes monitoring events into Interface-level mismatches, i.e., services with similar functionality, but through different WSDL interfaces and Protocol-level mismatches, i.e., mismatches concerning the order or number of supplied and required messages.

The proposed high-level event taxonomy [Zeginis et al. 2011] is based on two different criteria: (i) the affected SBA layer; and (ii) the service aspect concerned (functional, non-functional). The affected layer concerns the three functional layers analyzed in Section 1.2, while the service aspect concerns the service functional and non-functional characteristics [Papazoglou 2008]. The former ones detail the operational aspects that define the overall service behavior, such as the way and time it is invoked, while the latter concern quality attributes (e.g., response time and throughput). Its main advantage upon the other taxonomies is that it perfectly perfectly with the adopted SBA layers as well as the consideration of both functional and non-functional service aspects.

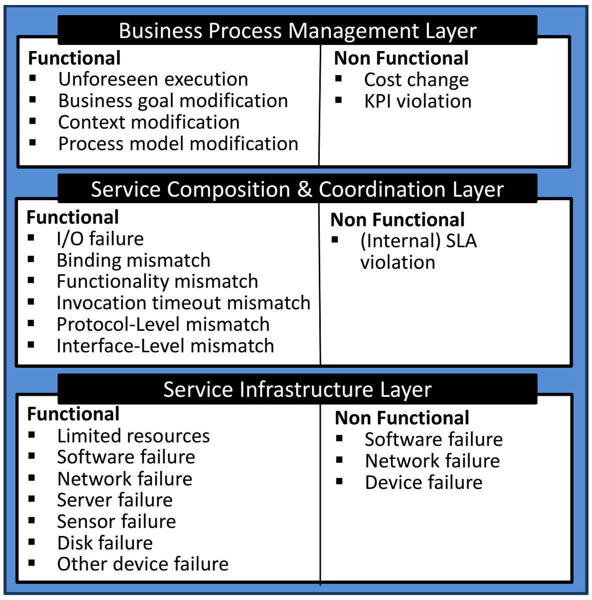


Figure 5.3: Taxonomy of SBA adaptation-related monitoring events

Figure 5.3 illustrates the proposed taxonomy of adaptation-related events for the three functional SBA layers. Indicatively, the BPM layer comprises mismatches regarding to the business process, such as KPI violations or monitoring events stemming from modifications at this layer (e.g., business goal or process model modifications). At the SCC layer, monitoring events focus on mismatches about service execution and QoS violations, such as I/O failures and SLA violations. Finally, at the SI layer, events mainly concern device failures affecting the overall SBA, such as limited resources or network failures.

5.2 Component Meta-Model

Along with our event meta-model presented in [Zeginis et al. 2013a], we propose a component meta-model to describe different source components for each event type. Our component meta-model (Figure 5.4) is useful for defining SBA system components and their direct interrelationships. To the best of our knowledge, there is no such extensible meta-model describing both the multi-Cloud SBA components and their adaptation possibilities.

The main benefits of this component meta-model are that it is extensive enough to capture the most common components of a multi-Cloud SBA, related to functional and non-functional violations, as well as it is extensible enough, in order to meet the needs of any SBA provider, which may incorporate other layer-specific components utilized by their applications. In addition, it can be exploited by any adaptation manager to design adaptation rules, based on the SBA components' properties, which have been carefully defined to stimulate the mapping from monitoring events to specific adaptation actions.

The main purpose of designing this meta-model is to capture the dependencies between components in a Multi-Cloud system, that can be further exploited to perform a root cause analysis for system faults. These dependencies of an instance component model are exploited by the Pattern Discovery Algorithm presented in Section 6.2 to detect valid patterns leading to critical SLO violations. In particular, component dependencies lead to selecting correct patterns in an event stream, where one event in the pattern leads to the next one. Furthermore, each one of the simple monitoring events described in an Event meta-model instance, maps to an instance of the Component class (i.e., the sourceComponent attribute of the Event Class) of the corresponding component model instance.

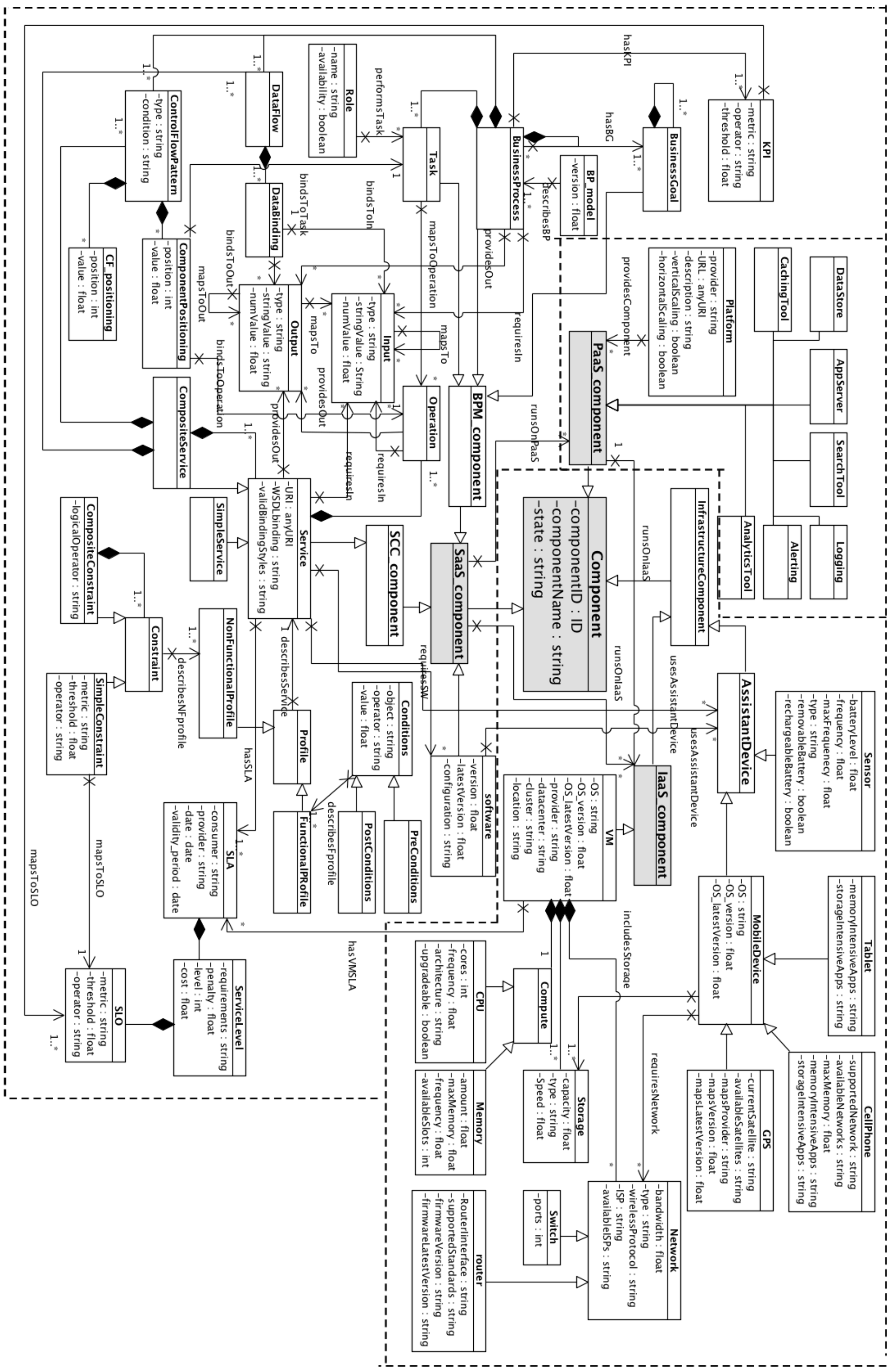


Figure 5.4: The component meta-model

5.2.1 Meta-Model Description

The main meta-model class is **Component**, characterized by a unique ID, its name and the state that it exposes. Components are further classified, based on the referenced layer, as **SaaS_component**, **PaaS_component**, or **IaaS_component**. Components at the SaaS layer can be further distinguished into **BPM_components** or **SCC_Components** and **Software_components**, depending on the SBA layer. Each subclass of the **Component** class has its own subclasses and interrelationships with other (sub)classes, providing a further specialization level which reflects the type of functionality exposed by that component. As far as the component interrelationships are concerned, we distinguish between generic associations of the superclasses (e.g. a SaaS component runs on an IaaS component) and concrete associations of the subclasses (e.g. a Service requires an Input). In this way, we can capture the component dependencies of the system, facilitating an SBA provider to design the instance component model of her/his multi-Cloud SBA.

As already analyzed in Section 3.1, there is already a couple of widely-used (standard and non-standard) models describing components of the Cloud layers (IaaS, PaaS and SaaS). Whenever designing a new (meta-)model it is desirable to reuse existing models, especially the standardized ones. Thus, in the proposed model we reuse the following parts of existing ones: (i) the modeling of a VM and its contents (i.e., Compute, Storage, Network) is inspired by the IaaS standard model proposed by OCCI (Figure 3.1), (ii) The PaaS modeling part borrows terms and classes from the CAMP and Cloud4SOA models (Figures 3.2 and 3.3 respectively); and (iii) the SCC and BPM components modeling is inspired by various models introduced in the S-Cube EU project. An extensive analysis of the model classes, their properties and their interrelationships can be found in Appendix B.

5.2.2 Traffic Management Component Model

In the same wavelength with subsection 5.1.2, here we extend the ongoing running example by defining the corresponding component model (bottom part of Figure 5.5) for the identified event pattern {we_1,ce_2,ce_4,we_3}. Figure 5.5 except from identifying the traffic management multi-Cloud application components and their dependencies, it also captures their distribution across the functional layers and the mapping from the source Components of the Event Model to the actual components of the Component Model. In particular, the following mappings are identified: (i) the event we_1 maps to the *CPU* component of the FlexiantVM class instance, (ii) the event ce_2 maps to the *Memory* component of the FlexiantVM class instance, (iii) the event ce_4 maps to the corresponding *NetworkUptime SLO*, (iv) the event we_3 maps to SLO defining the threshold for the execution time of the DeviceConfig service; and (v) the event E_7 maps to the KPI defining the threshold for the total duration of the Assess and DeviceConfig tasks. In the next chapter we will show how the underlying dependencies of these five components are utilized to validate event patterns extracted by the proposed pattern discovery technique.

5.3 Adaptation Actions Meta-Model

Until now we have identified an event meta-model for describing the monitoring events and their interrelationships through the specification of event patterns, and the adaptation rules (ECA triplets), as well as a component meta-model for the definition of the dependencies among the qualitative and quantitative attributes of an SBA and the monitored source components producing events at run-time. Thus, Events and Conditions of an ECA triplet are efficiently described via the event meta-model and the corresponding SLA documents. However, to the best of our knowledge, the Adaptation part is not sufficiently modeled for multi-Cloud SBA environments. There are already a couple of approaches [Blair et al.

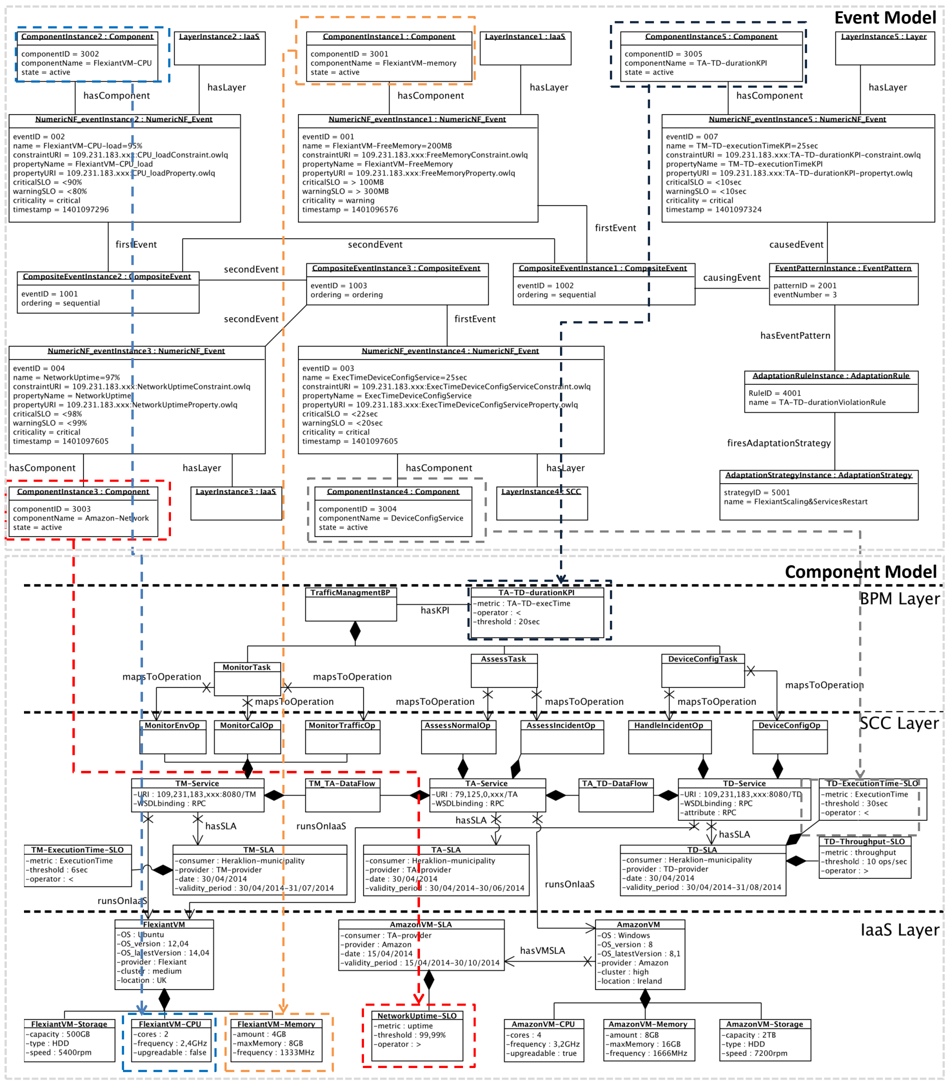


Figure 5.5: Mapping the traffic management application component model to the corresponding event model

2009, Inzinger et al. 2013, Marquezan et al. 2014] and domain specific languages (DSLs), such as CloudML[2], addressing adaptation-modeling issues, but they mainly provide abstract models for enacting adaptation at deployment time and do not take into consideration cross-layer aspects.

To this end, in this section we propose a meta-model for the adaptation ac-

[2]http://cloudml.org/

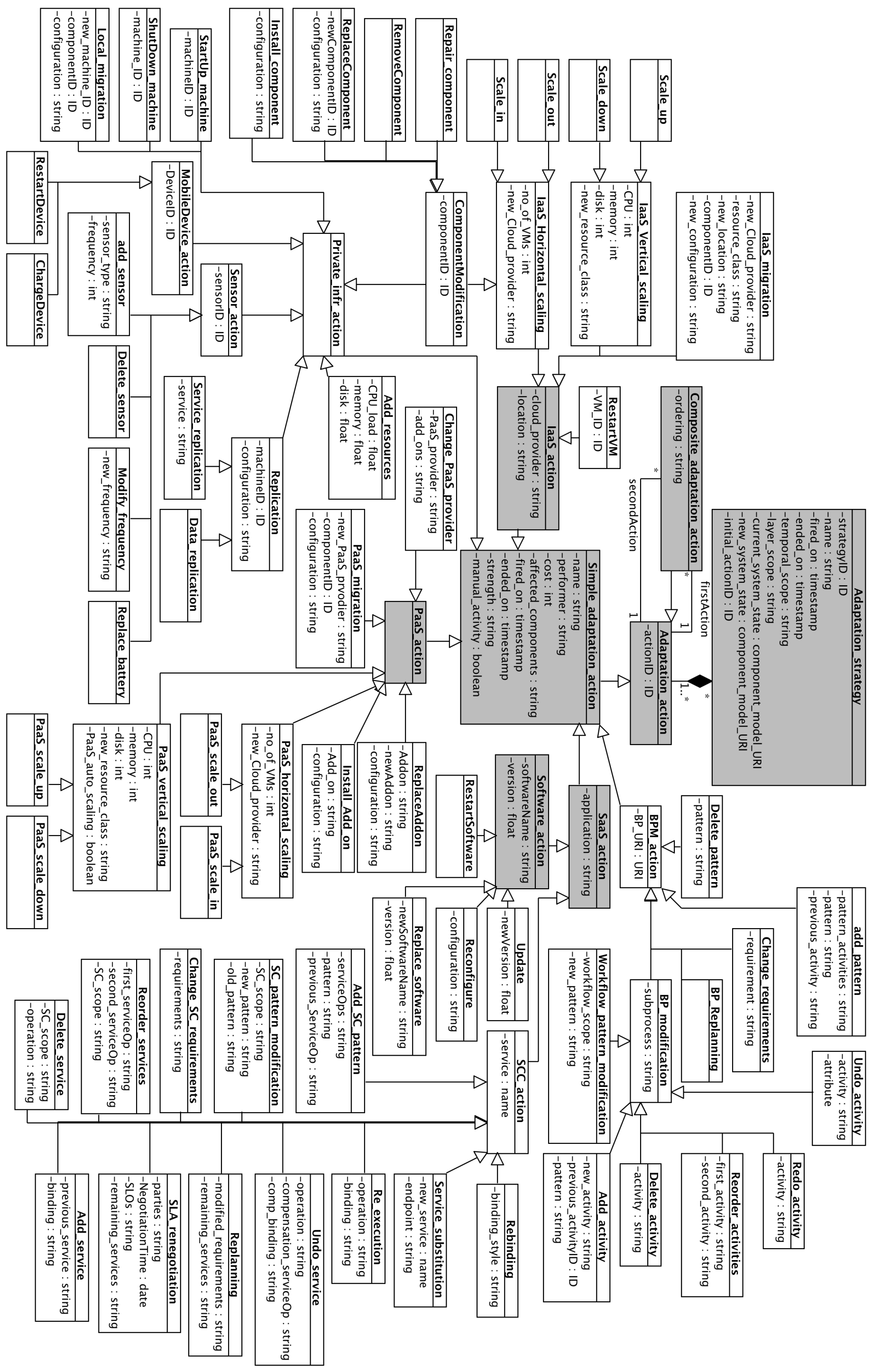


Figure 5.6: The adaptation actions meta-model

tions (Figure 5.6) that can be enacted on each one of the functional layers of an SBA. This meta-model's importance deserves, as it facilitates to a great extent the SBA Adaptation Manager of the proposed framework to perform the mapping from critical raw events to suitable adaptation actions, based on the instance of the adaptation action meta-model. In addition, it can be used to provide an overview of the framework's adaptation capabilities and of the adaptation history, including the enacted time of each adaptation strategy, its lower-level actions and the affected components.

5.3.1 Meta-Model Description

The main model class is **Adaptation_strategy** characterized by its name, duration, temporal and layer scope and the state it exposes. Each adaptation strategy comprises of a number of lower level **Adaptation_actions**, identified by its *actionID* and further classified as **Simple adaptation actions** and **Composite adaptation actions**, whether they consist of more than one simple action. Composite actions involves interrelated actions that should be performed together with the specified order (e.g., a **Redo_activity** BPM action implies a **Re-execution** SCC action). Simple adaptation actions are further classified, based on the referenced layer, as **SaaS_action**, **PaaS_action**, or **IaaS_action**. Components at the SaaS layer can be further distinguished into **BPM_actions** or **SCC_actions** and **Software_actions**, depending on the SBA layer. Similarly to the component meta-model, each subclass of the layer-specific super-classes has its own sub-classes, providing a further specialization level and the required configuration for each low-level action. In this way, we can capture all the available adaptation actions supported by our system, facilitating the Adaptation Manager during the adaptation/scalability rule derivation process. The semantics of the rest of the classes are straightforward, thus no detailed description is provided as a separate appendix.

5.3.2 Traffic Management Adaptation Actions Model

Continuing our ongoing example of the traffic management application, in this subsection we define an traffic management instance model of the previously proposed adaptation actions meta-model (bottom part of Figure 5.7). This figure additionally presents the mapping from the event meta-model instance, defining the adaptation rule for the identified event pattern {we_1,ce_2,ce_4,we_3}, to the specific adaptation strategy and its lower level adaptation actions. In particular, the *FlexiantScaling & ServiceRestart* adaptation strategy identified in the event model is further specialized in this figure into specific actions enacted in various functional layers. These actions also provide proper configuration for their successful invocation. In fact, this adaptation strategy is composed of three lower level events: (i) a simple **Scale_Up** action of the flexiantVM, upgrading it with quad-core CPU and 8GB RAM memory, i.e., from a "medium" to a "high" VM, (ii) a composite adaptation action comprising of two sequential simple adaptation actions; a **Redo_activity** of the Monitor Task and a **Re-execution** of the corresponding T_M service; and (iii) another composite adaptation action comprising of two sequential simple adaptation actions; a **Redo_activity** of the DeviceConfig Task and a **Re-execution** of the corresponding T_D service. In practice, this adaptation strategy is enacted before the actual E_7 event appears, thus enabling proactive adaptation. Similarly to the previous mappings, Figure 5.8 represents the mapping from the lower level actions to the affected components.

5.4 Conclusions

To sum up, in this chapter we have introduced three monitoring- and adaptation-aware meta-models, able to capture the monitoring and adaptation related aspects; (i) the monitoring events meta-model enables modeling the monitoring events, the formed event patterns, the adaptation ECA rules and the dependencies among the metrics through the specification of event patterns, (ii) the com-

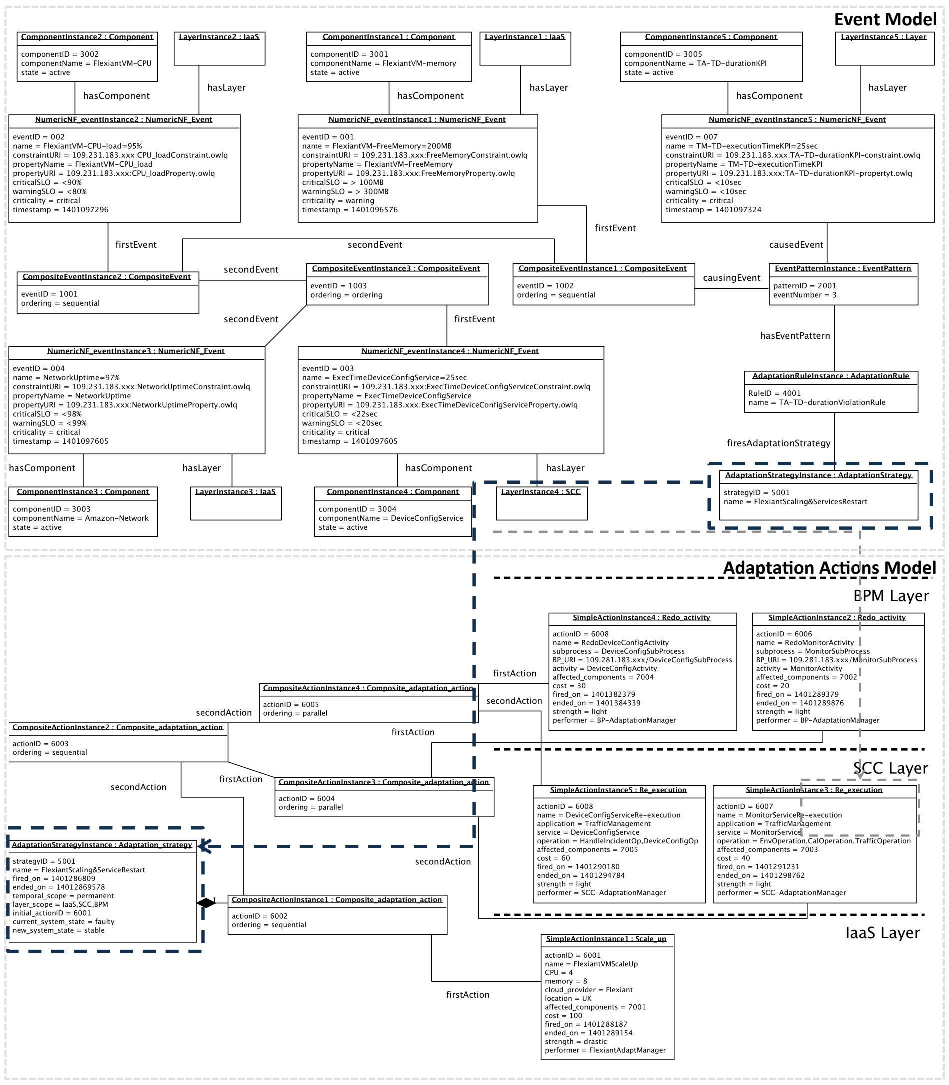

Event Model
ComponentInstance2 : Component
LayerInstance2 : IaaS
ComponentInstance1 : Component
LayerInstance1 : IaaS
ComponentInstance5 : Component
LayerInstance5 : Layer
hasComponent
hasLayer
NumericNF_eventInstance2 : NumericNF_Event
NumericNF_eventInstance1 : NumericNF_Event
NumericNF_eventInstance5 : NumericNF_Event
firstEvent
secondEvent
causedEvent
CompositeEventInstance2 : CompositeEvent
CompositeEventInstance3 : CompositeEvent
CompositeEventInstance1 : CompositeEvent
causingEvent
EventPatternInstance : EventPattern
NumericNF_eventInstance3 : NumericNF_Event
NumericNF_eventInstance4 : NumericNF_Event
hasEventPattern
AdaptationRuleInstance : AdaptationRule
firesAdaptationStrategy
AdaptationStrategyInstance : AdaptationStrategy
ComponentInstance3 : Component
LayerInstance3 : IaaS
ComponentInstance4 : Component
LayerInstance4 : SCC
Adaptation Actions Model
BPM Layer
SimpleActionInstance4 : Redo_activity
SimpleActionInstance2 : Redo_activity
firstAction
secondAction
CompositeActionInstance4 : Composite_adaptation_action
CompositeActionInstance2 : Composite_adaptation_action
CompositeActionInstance3 : Composite_adaptation_action
SCC Layer
SimpleActionInstance5 : Re_execution
SimpleActionInstance3 : Re_execution
AdaptationStrategyInstance : Adaptation_strategy
CompositeActionInstance1 : Composite_adaptation_action
IaaS Layer
SimpleActionInstance1 : Scale_up

Figure 5.7: Mapping the traffic management application event model to the corresponding adaptation actions model

ponent meta-model captures the layer-specific components of a multi-cloud application; and (iii) the adaptation actions meta-model identifies the adaptation strategies per Cloud and SOA layer. All these models are extensible and reusable by any interested Cloud SBA developer. We have also defined instances of these meta-models and the corresponding mappings, based on the traffic management example, in order to illustrate their applicability, usability and flexibility.

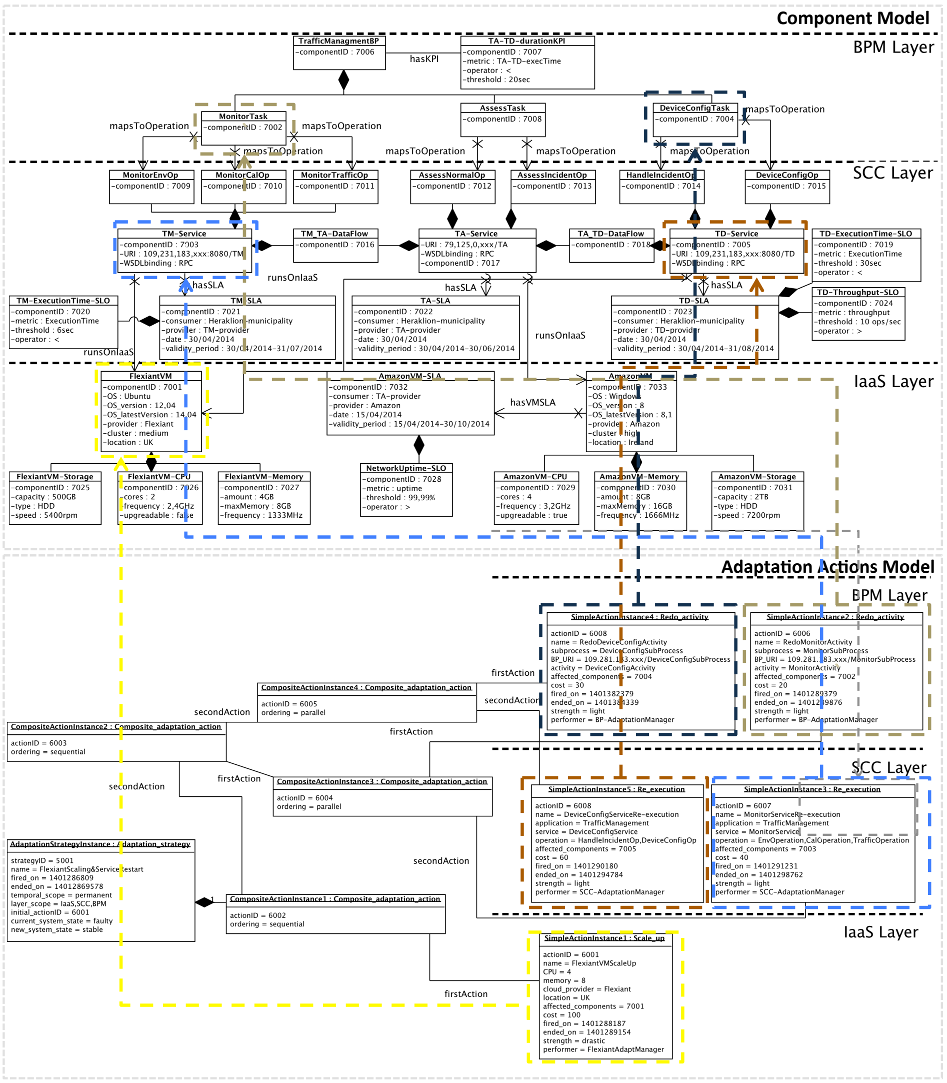


Figure 5.8: Mapping the traffic management application adaptation actions model to the corresponding component model

Chapter 6

Monitoring and Pattern Discovery

Contents

The literature review in Chapter 3 revealed that there is a research gap in cross-layer monitoring of multi-cloud SBAs. The main goal of this chapter is to present our contribution towards filling this gap. In addition, it focuses on the processing of the monitoring events to discover event patterns within an event stream of the SBA execution history, taking into consideration the component dependencies captured in the corresponding component model. In particular, Section 6.1 analyzes our proposed cross-layer monitoring approach and the underlying techniques utilized by the ECMAF's Monitor Manager. Section 6.2 proposes a logic-based pattern discovery algorithm and discussing its application on the traffic management example.

6.1 Cross-layer Monitoring

As discussed in Chapter 1, it is crucial to monitor all the functional layers of a SBA and effectively manage the monitoring events, so as to take the optimum decision for the most suitable adaptation action. These events should be synchronized and delivered in a correct order to the Adaptation Engine. The rationale behind the event synchronization and ordering is that a delivery delay may jeopardize the correct functioning of the Adaptation Engine.

In ECMAF we advocate for the use of independent monitoring components that can detect layer-specific events. Thus, in our framework we employ a set of monitoring mechanisms distributed in the deployed VMs, analyzed in Section 8.3.1. Then, the Monitor Manager should directly pass the events to the Adaptation Engine in the correct order. The incorporated time synchronization technique is in charge of keeping the logical clocks of the involved VMs synchronized. Moreover, when it comes to delivering events to the Adaptation Engine, we have to take into account the adaptation manager requirements. So, it is desirable to pass only the subscribed events, as some of the monitoring events may be useless for the adaptation manager. For instance, an Adaptation Manager that does not incorporate an infrastructure manager is not interested in any infrastructure events.

This section has a two-fold role: (i) it formally defines the semantics of the monitoring concepts of a multi-Cloud SBA; and (ii) it elaborates on how cross-layer monitoring is performed in the ECMAF framework, providing more details on the functionality of the Monitor Manager component. Figure 6.1 isolates the ECMAF's Monitoring Engine, in order to facilitate the reader in the next subsections, which frequently refer to the incorporated components.

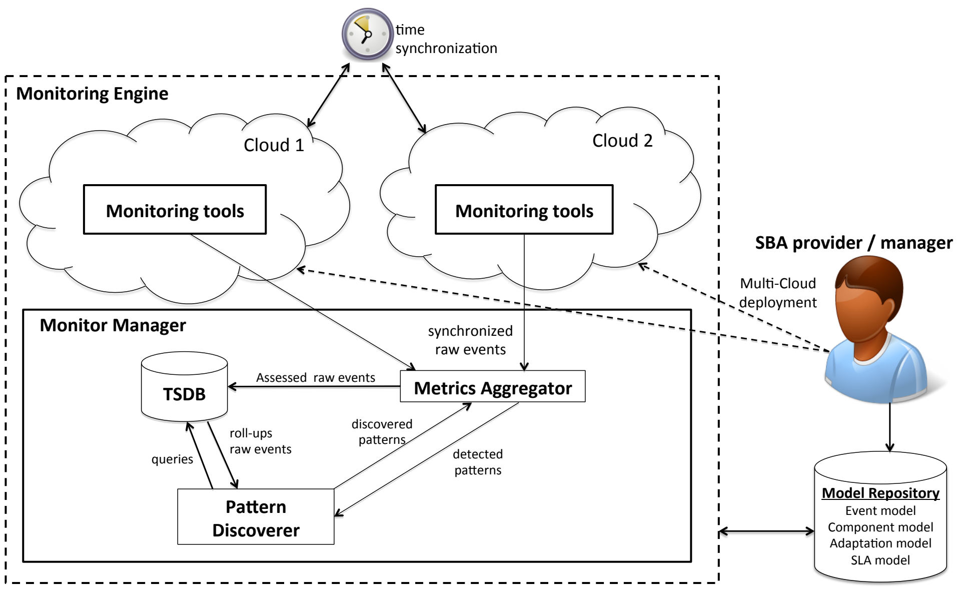


Figure 6.1: The ECMAF's monitoring engine

6.1.1 Metric Definition

In Section 3.4 we have presented OWL-Q [Kritikos and Plexousakis 2006], as the ECMAF's exploited QoS description model for Web services, that is able to define non-functional metrics, measuring various aspects of QoS-based Web service description. OWL-Q allows for describing metrics by identifying the QoS property and the service, on which they apply, the measurement directive for simple metrics, the computation formula for complex metrics, the metric type (simple, complex, dynamic, monotonic, etc.). In this section, we provide examples of OWL-Q metric definitions, either simple or composite, for the traffic management running example. Especially, Listings 6.1 and 6.2 provide OWL-Q definitions for a simple execution time metric and an average execution time complex metric respectively. The former one describes *DeviceConfig execution time*, i.e. metric e_3 (see Section 2.3). It is a dynamic, monotonic metric, acquiring positive values in the range of (0–10000). The latter one describes *Average execution time of Monitoring service* metric. It is a complex metric applying an *Average* OWL-Q function on the simple raw metric *Monitor_AvgExec*. In addition, it is also a dynamic, monotonic, QoS metric acquiring positive values in the range of (0–20000).

Listing 6.1: Simple metric (DeviceConfig service execution time) OWL-Q definition

```
<owlq-metric:SimpleMetric rdf:ID="DeviceConfigExec_Time">
  <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#IntervalMetric"/>
  <owlq-metric:hasScale rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Scale.owl#
       TimeIntervalScale"/>
  <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#ServiceMetric"/>
  <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#QoSMetric"/>
  <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#DynamicMetric"/>
  <owlq:hasValueType rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_ValueType.owl#
       Integer_0_10000"/>
  <owlq-metric:ofObject rdf:resource="#DeviceConfig"/>
  <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#PositiveMonotonicMetric"/
       >
  <owlq-metric:evaluates rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#
       ExecutionTime_1"/>
</owlq-metric:SimpleMetric>
```

Listing 6.2: Complex metric (Average execution time of Monitor service) OWL-Q definition

```
<owlq-metric:ComplexMetric rdf:ID="Monitor_AvgExec">
        <owlq-metric:evaluates>
          <owlq-function:MetricFunctionCall rdf:ID="Monitor_AvgExecTime">
            <owlq-function:callsFunction rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.
                 owl#Average"/>
            <owlq:hasArgument>
              <owlq-metric:QoSMetric rdf:ID="Monitor_ExecTime">
              <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#DynamicMetric"/
                   >
              <owlq-metric:hasScale rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Scale.owl#
                   TimeIntervalScale"/>
              <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#ServiceMetric"/
                   >
              <owlq-metric:ofObject rdf:resource="#MonitorService"/>
              <owlq:hasValueType rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_ValueType.owl#
                   Integer_0_20000"/>
              <owlq-metric:evaluates rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#
                   ExecutionTime_1"/>
              <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#IntervalMetric"
                   />
              <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#
                   PositiveMonotonicMetric"/>
              <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#SimpleMetric"/>
            </owlq-metric:QoSMetric>
      </owlq:hasArgument>
   </owlq-function:MetricFunctionCall>
   </owlq-metric:evaluates>
   <owlq-metric:derivedFrom rdf:resource="#MonitorServiceExec_Time"/>
   <owlq-metric:ofObject rdf:resource="#MonitorService"/>
   <owlq:hasValueType rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_ValueType.owl#Integer_0_100"
        />
   <owlq-metric:hasScale rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Scale.owl#
        TimeIntervalScale"/>
   <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#QoSMetric"/>
   <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#IntervalMetric"/>
   <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#NegativeMonotonicMetric"/>
   <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#DynamicMetric"/>
   <rdf:type rdf:resource="http://127.0.0.1:50000/OWL-Q/OWL-Q_Metric.owl#ServiceMetric"/>
</owlq-metric:ComplexMetric>
```

6.1.2 Event Representation

In the previous section we have formally defined the definition of metrics in OWL-Q language. In this section we provide the annotations for the metric's event representation. As already analyzed in Section 5.1, each captured event has a number of properties characterizing it. Thus, a monitoring event is a multi-attribute item, uniquely identified by its *EventID*. More formally, given a *n*-attribute event stream *D* with attributes $A = \{A_1, A_2, \dots, A_n\}$, each event $e \in D$ is a n-tuple with the following required attributes, that should be provided by the monitoring engine, except the criticality attribute which is determined during the SLO assessment process.

- **EventID:** This attribute uniquely characterizes the event / measurement. In practice, it is an incremental counter of the Metrics Aggregator component assigning IDs to the collected events, as soon as they reach the Monitor Manager.

- **Metric:** This attribute identifies the corresponding metric of the considered event. For instance, a measurement maps to the *Temperature* metric of the Environmental Monitor task of the traffic management running example.

- **Value:** This attribute identifies the value, either numeric or textual that would be assessed by the Metrics Aggregator, in order to detect SLO violations, in accordance to the SLA thresholds defined in the corresponding WSLA document.

- **Timestamp:** This attribute identifies the exact time when this event / measurement is detected by a monitoring engine. In practice it is defined in *Unix time* (i.e., the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970,) and as already pinpointed all monitoring engines are timely synchronized (see Section 4.2.2 for more details).

- **Layer:** As our approach is cross-layer, with a main purpose to identify patterns of monitoring events that may dispersed across various functional SOA and Cloud layers. Thus, we employ this attribute to directly refer to the layer, where the event / measurement is detected.

- **Application:** The Metrics Aggregator component may collect events from various applications, thus this attribute is necessary to discern them using this attribute.

- **Task:** This attribute identifies the application's task (i.e., SCC and BPM events) related to the monitoring event / measurement (e.g., the assessment task of the traffic management application).

- **ComponentID:** This attribute identifies the exact component that produced the monitoring event, as it is defined the SBA's component model (e.g., component with ID=7027 is the FlexiantVM memory in our running example).

- **HostID:** This attribute determines the VM or private machine that produced the monitoring event, independently of the related functional layer.

- **Criticality** This attribute is defined after the assessment process performed by the Metrics Aggregator component and acquires one of the following values: *success, warning* or *critical*, as they are defined in the next subsection.

In order to ensure that the required information is passed from the individual monitoring components, we have designed the corresponding XML schema (i.e.

XSD)[1]. Monitoring events are sent to the Metrics Aggregator component, either in a XML or a JSON format and are validated according to these schemas. If they are not valid, i.e. either all the required fields are not filled or an attribute type is not correct, they are not taken into account for the pattern discovery process and they are not stored in the TSDB. These monitoring engines should be checked in order to verify the cause of the invalid event files. The following Listings 6.3 and 6.4 define sample monitoring events of the traffic management example, in XML and JSON format respectively.

Listing 6.3: An XML file representing a warning event of DeviceConfig Service execution time (event e_3) metric

```xml
<?xml version="1.0"?>
<NumericNF-event>
  <eventID> 003 </eventID>
  <name> DeviceConfig ExecTime SLA warning Violation </name>
  <application> TrafficManagementApp </application>
  <task> DeviceConfigTask </task>
  <hostID> 7001 </hostID>
  <propertyName> DeviceConfigExecTime </propertyName>
  <hasLayer> SCC </hasLayer>
  <detectedNumericValue> 21 </detectedNumericValue>
  <hasComponent> 7019 </hasComponent>
  <timestamp> 1401097605 </timestamp>
  <criticality> warning </criticality>
  <constraintURI> 109.231.183.xxx:ExecTimeDeviceConfigExecTimeConstraint.owlq </constraintURI>
  <warningSLO> DeviceConfigExecTime<20sec </warningSLO>
  <criticalSLO> DeviceConfigExecTime<22sec </criticalSLO>
</NumericNF-event>
```

Listing 6.4: A JSON file representing a critical event of FlexiantVM's CPU-load (event e_2) metric

```json
{
    "NumericNF-Event": {
        "eventID": "002",
                "name": "FlexiantVM CPU-load critical violation,
                "application": "TrafficManagementApp",
                "hostID": "7001",
                "propertyName": "FlexiantVm-CPU-load",
                "detectedNumericValue": "92%",
                "hasLayer": "IaaS",
                "hasComponent": "3002",
```

[1]www.ics.forth.gr/ zegchris/schemas

```
                "timestamp": "1401097296",
                "criticality": "critical",
                "constraintURI":"109.231.183.xxx:CPU-loadConstraing.owlq",
                "warningSLO": "<80%",
                "criticalSLO": "<90%"
    }
}
```

Monitoring Event Types

Before giving details on how cross-layer monitoring is performed, we define the three criticality types of the monitoring events:

- **Successful events**. These events carry information about successful invocations (correct input/output) and normal state of the system components. For example, a successful ping message from a server that hosts a service of our SBA indicates that this server is running normally. This type of events are ignored, as they do no provide information about improper execution of the monitored SBA.
- **Warning events**. These events indicate that a component (device, service, software, etc) of our system does not perform normally and a monitoring property has exceeded the warning threshold, as it is defined in the SLA document. For example, the Monitoring Engine reports a warning event when the execution time of the Assessment service is 9ms, while the warning threshold is 8ms and the critical threshold is 10ms in the normal traffic cases. These messages warn the SBA manager that something wrong is happening at a specific component and help us to proactively adapt the faulted component before this malfunction deteriorates.
- **Critical events**. Events of this type indicate a failure during the SBA execution. These events are detected by our cross-layer monitoring mechanism when a monitoring property is violated, i.e. when it exceeds the critical threshold defined in the SLA document. For example, a critical event can

be the low average availability of the Monitoring Web service below 99% after a couple of invocations. Moreover, critical events, as well as warning events, can capture functional properties, as for example the correct input type of a specific service or the low available memory of a VM.

6.1.3 Event Processing

As already analyzed in Section 4.2 the main functionality of the Metrics Aggregator component is to collect the monitoring events / measurements and process them before storing them in a Time-series Database (TSDB). The following two subsections elaborate on the assessment and storage of the monitoring events.

Monitoring Events' Assessment

As soon as a monitoring event reaches the Metrics Aggregator component, it is assessed according to the constraints defined as SLO objectives in the agreed SLA document. Upon the SLO assessment against the corresponding WSLA document, the criticality of the assessed event / measurement is filled in the corresponding XML or JSON file and then stored in the TSDB. In addition, if the assessment process indicates a critical violation then it directly triggers the most appropriate adaptation strategy, as it has already been determined by the application expert. Otherwise, if the monitoring event maps to a composite metric, the Metrics Aggregator queries the TSDB to get the average value of the raw metrics monitored in the interval defined in the metric's definition, i.e., the complex metric's OWL-Q file. Listing 6.5 represents a part of the traffic management applications' WSLA document defining the criticality of the *DeviceConfigExecTime* event collocated in Listing 6.3. It defines a critical SLO for the corresponding metric. In particular, the measurements of the DeviceCofigExecTime metric should be less than 22sec when the FlexiantCPU-load is less than 90%. This SLO is valid for a 7-month period and the measurements are assessed whenever a new value is available (i.e. expression *<EvaluationEvent>NewValue</EvaluationEvent>*).

Listing 6.5: Sample SLO for the traffic management application

```
<ServiceLevelObjective name="DeviceConfigExecTimeCritical">
  <Obliged>AppProvider</Obliged>
  <Validity>
        <Start>2014-06-01:00:00.00</Start>
        <End>2014-12-31:00:00.00</End>
  </Validity>
  <Expression>
        <Implies>
                <Expression>
                        <Predicate xsi:type="Less">
                                <SLAParameter>DeviceConfigExecTime</SLAParameter>
                                <Value>22</Value>
                        </Predicate>
                </Expression>
                <Expression>
                        <Predicate xsi:type="Less">
                                <SLAParameter>FlexiantVM-CPU-load</SLAParameter>
                                <Value>0.9</Value>     <!-- 90% -->
                        </Predicate>
                </Expression>
        </Implies>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Monitoring Events' Storage

As already discussed in Section 3.4, TSDBs are the ideal solution when it comes to storing timestamped data. Apart from the efficient storage (accomplished by Apache Cassandra database[2]), the selected KairosDB TSDB (see more technical and architectural details in Section 8.3.1) additionally provides roll-up capabilities, i.e. aggregation functions (e.g. average, min, max, etc.) across user-defined time ranges and even lets you combine aggregators. In addition, except from the default field for event storage, i.e. timestamp, metric and value, it allows data points' labeling with additional tags, consisting of a name and value. Thus, the rest event attributes (eventID, name, application, task, componentID, layer, criticality) are defined by separate tags, in order to facilitate the filtering process utilized by the pattern discovery algorithm analyzed later in this chapter. Listing 6.6 represents a data point structure, as it is stored in the underlying Cassandra DB.

[2]http://cassandra.apache.org/

Listing 6.6: A sample TSDB data point

```
[{
    "name": "FlexiantVM-CPU-load",
    "timestamp": 1401097296,
    "value": 0.92,
    "tags":{"hostID":"7001",
                    "eventID": "002",
                        "application": "TrafficManagementApp",
                        "Layer": "IaaS",
                        "ComponentID": "3002",
                        "criticality": "critical"
                   }
}]
```

6.2 Pattern Discovery

In Chapter 1 we have argued about the need and significance of interrelating the monitoring events and performing a root cause analysis to determine the main cause of failures in multi-Cloud deployments. This section analyzes one of the main contributions of this thesis; a novel pattern discovery technique enabling event correlation exploiting the source components' dependencies. The benefits of this method are twofold, also allowing the identification of new component dependencies, except the ones initially determined by the application manager and deriving metric dependency trees (MDTs) correlating the application's monitored properties. The following subsections elaborate on this technique and exemplify its application on the running example.

6.2.1 Pattern Discovery Algorithm

This section presents an offline deterministic algorithm (Algorithm 1) for discovering frequent event patterns (i.e. association rules) leading to critical events of a specific metric. This algorithm is based on propositional logic [Sim et al. 2010] and takes into consideration both the dependencies of the components constituting the multi-Cloud application system and the aggregate metrics extracted by the TSDB. In particular, the algorithm requires as input: (i) the monitoring events stream, (ii) the application and (iii) the metric we are interested in identifying critical event patterns, (iv) the metrics classification (i.e. success, warning, critical), that is part of the aggregated event for the specified intervals; and (v) an instance of the monitored SBA component model. The algorithm exploits contingency tables (see Table 3.2), i.e. tables displaying the frequency distributions of the candidate patterns and their negations as antecedents and the specified metric event, as well as its negation as consequences. The entries of these tables (see Figure 6.2) are utilized to determine the association rules.

Algorithm's Description

The aim of the proposed event pattern discovery algorithm is to discover frequent patterns leading to violations of specific aggregate metrics. We assume that only warning or critical events lead to other critical events. Therefore, a filtering of the event stream (*line 3*) is performed before the actual discovery process starts. The filtering does do not consider events coming by different Cloud providers from the ones used by the monitored application or by other Cloud providers in single Cloud deployments. Then, the event stream is split into user-defined time intervals (*line 4*), which are actually the intervals of the aggregate metric being assessed. Thus, one should carefully define the aggregate metric's measurement interval, in order to be able to catch event patterns, as accurately as possible.

As far as the critical intervals are concerned (i.e. the corresponding average value indicates a violation), we calculate the temporal ordered powersets of the raw events subset, from the first interval event to the event before the last critical raw measurement (see Figure 6.3), as candidate patterns and then update the powersets' tree (Figure 6.2). Concerning non-critical intervals, we also calculate the powerset of the whole interval, as candidate patterns not leading to violations of the specified metric. Each powerset is filtered based on the current component model dependencies (see an example in Section 6.2.3). If a powerset's set does not

Algorithm 1 Event pattern discovery algorithm

```
1:  Input: event stream, application, metric, interval, metrics classification, component model
2:  Output: pattern graph, pattern ranking, coherent rules, ambiguous rules
3:  Filter raw events (ignore success /other applications'/other Cloud provider events)
4:  Divide event stream in event time intervals
5:  Calculate average metric value for each interval
6:  while not end of event stream do
7:      A = events before the last critical raw event in this interval
8:      $I_A$ = P(A) $\rightarrow$ temporal ordered power sets of set A
9:      filter power sets according to component model
10:     update power sets tree
11: end while
12: for $i \leftarrow 1, treelevels$ do
13:     for $j \leftarrow 1, treebranches$ do
14:         $\mathbf{B_{i,j}}$ = current branch
15:         while not end of event stream do
16:             A = events before the last critical raw event in this interval
17:             C = critical raw event for the specified metric
18:             compute S($B_{i,j}$, C) in A
19:             compute S($B_{i,j}$, $\neg C$) in A
20:             compute S($\neg B_{i,j}$, C) in A
21:             compute S($\neg B_{i,j}$, $\neg C$) in A
22:             update contingency table
23:         end while
24:         if S($B_{i,j}$, C) $>$ S($B_{i,j}$, $\neg C$) and S($B_{i,j}$, C) $>$ S($\neg B_{i,j}$, C) and S($\neg B_{i,j}$, $\neg C$) $>$ S($B_{i,j}$, $\neg C$) and S($\neg B_{i,j}$, $\neg C$) $>$ S($\neg B_{i,j}$, C) then
25:             create association rule ($B_{i,j} \rightarrow C$)
26:             compute $\lambda$
```

```
27:             store $(B_{i,j} \to C)$ in association rules Knowledge Base
28:         **else if** $(S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C) = (S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C))$ **then**
29:             Store $(B_{i,j} \to C)$ as **ambiguous rule**
30:         **else**
31:             compute $\lambda'$
32:         **end if**
33:     **end for**
34: **end for**
35: traverse tree
36: rank patterns on levels based on their $\lambda$ scores
37: filter discarded sets based on their $\lambda'$ scores
```

contain events, the source components of which are interrelated with each other, according to the instance of the component model, then this set is not further processed as a candidate pattern (*lines 6-11*). However, the algorithm considers the single events that might map to the critical violation. It is worth mentioning that the powersets' tree stores unique sets and there are no duplicates.

Up to this point, all the candidate patterns are stored in a tree-based structure (Figure 6.2). Then, a level-based and a branch-based tree traversal are performed in order to calculate the frequencies to be stored in each contingency table entry. Note that every discovered pattern reserves its own contingency table. The following frequencies are calculated (*lines 15-23*), where $B_{i,j}$ is the concerned sub-branch (*i* indicates the tree level, while *j* the branch counter) and C represents the critical aggregate event (i.e. violation) of the specified metric (see Figure 6.2). The negation of a pattern means that either of the included events does not appear. More formally, for the discovered pattern $\{e_1, e_2, e_4\}$, the following equivalence holds:

$$\neg\{e_1, e_2, e_4\} \equiv \neg e_1 \vee \neg e_2 \vee \neg e_4 \tag{6.1}$$

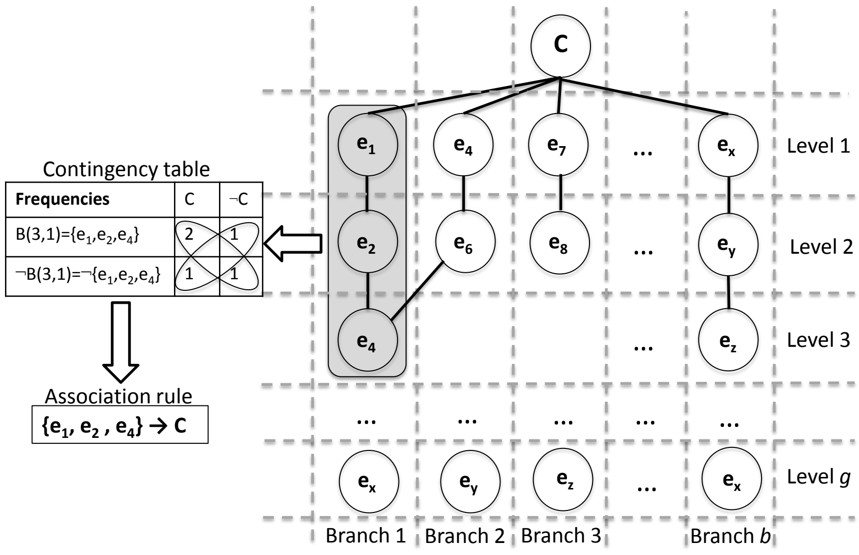


Figure 6.2: Powersets tree

- $S(B_{i,j}, C)$ is the frequency of $B_{i,j}$ set in critical intervals.
- $S(\neg B_{i,j}, \neg C)$ is the frequency of $\neg B_{i,j}$ set in non-critical intervals.
- $S(\neg B_{i,j}, C)$ is the frequency of $\neg B_{i,j}$ set in critical intervals.
- $S(B_{i,j}, \neg C)$ is the frequency of $B_{i,j}$ set in non-critical intervals.

In critical intervals the considered aggregate metric is violated, while in non-critical ones, it is not violated. The frequencies utilized to determine an association rule are: (i) $S(B_{i,j}, C)$; and (ii) $S(\neg B_{i,j}, \neg C)$. The first one obviously proves an association rule, that is under consideration, while the second one is considered to determine an association rule, as the concurrent absence of a pattern and the event indicating a violation of a composite event (i.e. the average value of the specified metric in an interval) also interrelates the root with the cause of a SLO violation. Consequently, in order to determine an association rule, the following relationship must be met (*lines 24–25*):

$$(S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C) > (S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C)) \qquad (6.2)$$

This equation means that in order to determine an association rule, the sum of the frequencies proving it should be greater than the sum of the frequencies weakening it. In the extreme case of equality (*lines 28–29*), an *ambiguous rule* is defined and this case should be reconsidered in another execution of the algorithm on a different sample dataset. In any other case, no association rule is discovered, but even in this case, a confidence level indicating the weakness of the discarded set is taken into account, so as to reconsider rules with low confidence level.

The confidence level λ (*line 26*) of the determined association rules, indicating the power of the rule, is calculated by the following formula. Higher λ scores mean stronger association rules:

$$\lambda = \frac{((S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C)) - ((S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C))}{S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C) + S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C)} \tag{6.3}$$

The confidence level λ' (*line 31*) of the discarded sets, indicating the irrelevance of the involved events, is calculated as follows. Higher λ' scores mean more irrelevant power sets:

$$\lambda' = \left| \frac{((S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C)) - ((S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C))}{S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C) + S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C)} \right| \tag{6.4}$$

In this context a *pattern* is defined as follows:

Definition 2 *Pattern is an association rule with confidence level* $\lambda > 0$*.*

However, the λ and λ' scores are highly dependent on the size of the dataset used for the pattern discovery, as well as on the iterations used so far to compute these scores. Consequently, we normalize these two scores using these two parameters (i.e. dataset size and number of algorithm iterations) for the second and so on iterations of the algorithm. Therefore, for the first iteration we use Equations 6.3 and 6.4 to calculate the λ and λ' scores, while for every next iteration *k*, Equation 6.5 is used to update these scores for the already discovered pattern p_i. *k* represents the number of algorithm's iterations that have discovered the pattern p_i, n_k is the dataset size used for the *k*th iteration of the algorithm and $\bar{n}$ is the average size of the sample dataset used so far to calculate the two scores of the pattern p_i. Equation 6.5 utilizes Equations 6.6 and 6.7 to calculate the updated λ scores.

$$\lambda_{(p_i,k)} = \bar{w} * \bar{\lambda} + w_k * \lambda_k \tag{6.5}$$

$$\bar{w} = \left\{ \begin{array}{ll} \frac{k-1}{k} * (1 - \frac{n_k - \bar{n}}{n_k}) & : n_k >= \bar{n} \\ \frac{k-1}{k} * \frac{\bar{n} - n_k}{\bar{n}} & : n_k < \bar{n} \end{array} \right\} \tag{6.6}$$

$$w_k = \left\{ \begin{array}{ll} \frac{1}{k} * \frac{\bar{n} - n_k}{\bar{n}} & : n_k >= \bar{n} \\ \frac{1}{k} * (1 - \frac{n_k - \bar{n}}{n_k}) & : n_k < \bar{n} \end{array} \right\} \tag{6.7}$$

Finally, the discovered association rules are ranked on a 5-level scale based on their λ scores (*lines 35–36*) as indicated in Table 6.1. Rules with high confidence level (i.e. strong and very strong rules) are given higher priority during the pattern detection process, immediately triggering the corresponding adaptation rules (enabling proactive adaptation), while looser rules (i.e. moderate, loose and very loose rules) are fired only after a couple of occurrences of the whole pattern and not upon the detection of a sub-pattern (as it happens with the strong rules). Going back to our running example, if the pattern {e_1, e_2, e_4} with λ=0.2 is detected (see Section 6.2.3), no rule is fired directly, unless the corresponding critical event (i.e. a violation of the Monitoring service execution time) occurs. On the contrary, if a very strong pattern {e_4, e_6} with λ=0.6 is detected, then the rule is immediately fired to prevent an execution time violation of the DeviceConfig service.

Discarded rules are also ranked (*line 37*) based on their λ' scores. Rules with high λ' scores ($\lambda' \in$ (0,4–1]) are stored as not relevant rules, so as to be omitted during every next execution of the algorithm, thus limiting the pattern search domain. Rules with low λ' scores ($\lambda' \in$ (0–0,2]) are simply ignored and will be reconsidered in other executions.

Table 6.1: Association rules ranking

λ range	Ranking	Rule strength
(0–0,2)	1	very loose
[0,2–0,4)	2	loose
[0,4–0,6)	3	moderate
[0,6–0,8)	4	strong
[0,8–1]	5	very strong

Algorithm's Complexity

As far as the algorithm's complexity is concerned, it depends on the size of the event stream *n*, the number of levels of the tree *g* and the number of tree branches *b*. The last two values, i.e. *b* and *g* are inversely proportional, as while increasing the patterns' length (i.e. *g* value), less unique patterns are discovered (i.e. *b* value). Consequently, as the algorithm requires *g*b* (i.e. the powersets' cardinality) iterations on the event stream, the algorithm's complexity is $\mathcal{O}(n*b*g)$. This complexity means that an extensive traversal of the powersets' tree is performed to find the association rules, their confidence level λ scores and finally rank them based on these scores.

6.2.2 Enhanced Pattern Discovery Algorithm

In the previous section we have proposed a novel single-pass and deterministic pattern discovery algorithm for identifying event patterns causing specific SLO violations. This algorithm presented in [Zeginis et al. 2014a] has been extended and enhanced in order to optimize its time and space complexity. The main difference of this enhanced algorithm is that it is a multi-pass and non-deterministic algorithm taking into account the results and rankings of the considered sets in order to identify only the new association rules.

Before introducing this enhanced algorithm, we introduce the types of sets' notations. In particular, as it is required to exploit the already discovered, discarded and ambiguous association rules, we define the following symbols for these rule sets:

- $\mathcal{D}$: the set of already discovered association rules
- $\mathcal{A}$: the set of ambiguous association rules (λ=0) discovered in previous algorithm's iterations
- $\mathcal{B}$: the set of discarded association rules with low λ' scores identified in previous algorithm's iterations

This updated algorithm (Algorithm 2), taking into consideration the results from previous executions, reduces to a great extent the candidate patterns, as we omit the already discovered and the discarded patterns. Thus, the total number of sets $\mathcal{N}$, considered as candidate patterns, is:

$$\mathcal{N} = |P(A)| - |\mathcal{D} \cup \mathcal{B}| + |\mathcal{A}| \tag{6.8}$$

6.2.3 Pattern Discovery on the Running Example

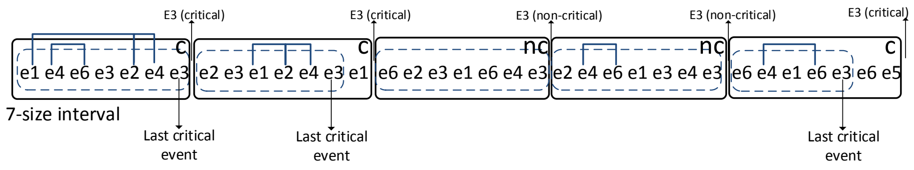


Figure 6.3: Event stream split – pattern discovery algorithm

Figure 6.3 clarifies the process of splitting the event stream and the way the intervals are further processed to identify patterns. Suppose that we are interested in finding patterns leading to violations of the average DeviceConfig service execution time (i.e. violation of the E_3 event). This event stream consists of 35 events

Algorithm 2 Enhanced Event pattern discovery algorithm

```
1: Input: event stream, application, metric, interval, metrics classification, component model, set $\mathcal{D}$, set $\mathcal{A}$, set $\mathcal{D}$
2: Output: pattern graph, pattern ranking, coherent rules, ambiguous rules
3: Filter raw events (ignore success /other applications'/other Cloud provider events)
4: Divide event stream in event time intervals
5: Calculate average metric value for each interval
6: while not end of event stream do
7:     A = events before the last critical raw event in this interval
8:     $I_A$ = P(A) $\rightarrow$ temporal ordered power sets of set A
9:     filter power sets according to component model
10:    update power sets tree
11: end while
12: for $i \leftarrow 1, treelevels$ do
13:    for $j \leftarrow 1, treebranches$ do
14:        $\mathbf{B_{i,j}}$ = current branch
15:        if $B_{i,j} \notin (\mathcal{D} \cup \mathcal{B})$ then
16:            while not end of event stream do
17:                A = events before the last critical raw event in this interval
18:                C = critical raw event for the specified metric
19:                compute S($B_{i,j}$, C) in A
20:                compute S($B_{i,j}$, $\neg C$) in A
21:                compute S($\neg B_{i,j}$, C) in A
22:                compute S($\neg B_{i,j}$, $\neg C$) in A
23:                update contingency table
24:            end while
25:            if S($B_{i,j}$, C) $>$ S($B_{i,j}$, $\neg C$) and S($B_{i,j}$, C) $>$ S($\neg B_{i,j}$, C) and S($\neg B_{i,j}$, $\neg C$) $>$ S($B_{i,j}$, $\neg C$) and S($\neg B_{i,j}$, $\neg C$) $>$ S($\neg B_{i,j}$, C) then
```

```
26:                create **association rule** $(B_{i,j} \rightarrow C)$
27:                compute $\lambda$
28:                store $(B_{i,j} \rightarrow C)$ in association rules Knowledge Base
29:            **else if** $(S(B_{i,j}, C) + S(\neg B_{i,j}, \neg C) = (S(\neg B_{i,j}, C) + S(B_{i,j}, \neg C))$ **then**
30:                Store $(B_{i,j} \rightarrow C)$ as **ambiguous rule**
31:            **else**
32:                compute $\lambda'$
33:            **end if**
34:        **end if**
35:    **end for**
36: **end for**
37: traverse tree
38: rank patterns on levels based on their $\lambda$ scores
39: filter discarded sets based on their $\lambda'$ scores
```

(after event filtering) (Figure 6.3) of the 7 different event types presented in Section 2.3. Each interval's power sets are processed to determine association rules (i.e. the connected events in the figure).

Thus, the event stream, containing raw events/measurements, is split into five intervals according to the metric E_3 definition. Three intervals are critical (*c* mark on the upper right corner of the interval) and two are non-critical (*nc* mark on the upper right corner of the interval). For the critical intervals we consider the subset before the last critical event for the DeviceConfig service execution time, while for the non-critical ones we consider the whole interval. The algorithm discovers two patterns and extracts the corresponding association rules, i.e. (i) $\{e_1, e_2, e_4 \rightarrow E3\}$ and (ii) $\{e_4, e_6 \rightarrow E3\}$, where E3 represents a violation of the average DeviceConfig service execution time. The contingency tables with the corresponding λ scores are shown in Table 6.2. This example involves only a couple of monitoring events and two discovered event patterns. However, dur-

Table 6.2: Contingency tables for the discovered patterns

Frequency	Y = E3	Y = $\neg E3$
$X = \{e_1, e_2, e_4\}$	2	1
$\neg X = \neg \{e_1, e_2, e_4\}$	1	1
λ = 0,2		

Frequency	Y = E3	Y = $\neg E3$
$X = \{e_4, e_6\}$	2	0
$\neg X = \neg \{e_4, e_6\}$	1	2
λ = 0,6		

ing the execution of real-life SBAs deployed on multiple Clouds, various events are detected and thus there are plenty of discovered event patterns.

Figure 6.4 depicts the component model of the Traffic Management Application capturing the system components' main dependencies, utilized by the proposed pattern discovery algorithm to validate the discovered event patterns. Concerning the two discovered patterns, the source components of the involved metrics (i.e. Flexiant VM's memory and CPU and Amazon VM's network Uptime for pattern (i) and Amazon VM's Network Uptime and Monitor SaaS's Execution time for pattern (ii) are interrelated via the common dataflows of the considered services. In order to check the validity of the components' dependencies, we exploit the underlying MySQL database storing information about the application components' state and interrelationships. For instance, in order to check if the produced events e_1 and e_2 are interrelated, we execute the following SQL query (Listing 6.7). This query returns the total number of active components included in the VM hosting the Monitoring and Device Configuration services (componentID=7001) and having ID 7026 or 7027 (i.e., the components producing e_1 and e_2 events), in order to identify if both components reside in the same VM and thus affect each other when a violation occurs. Many other SQL queries are incorporated in the system, validating every single dependency among system components. These SQL queries are designed by the Adaptation Manager provided that the SBA designer provides the corresponding component model.

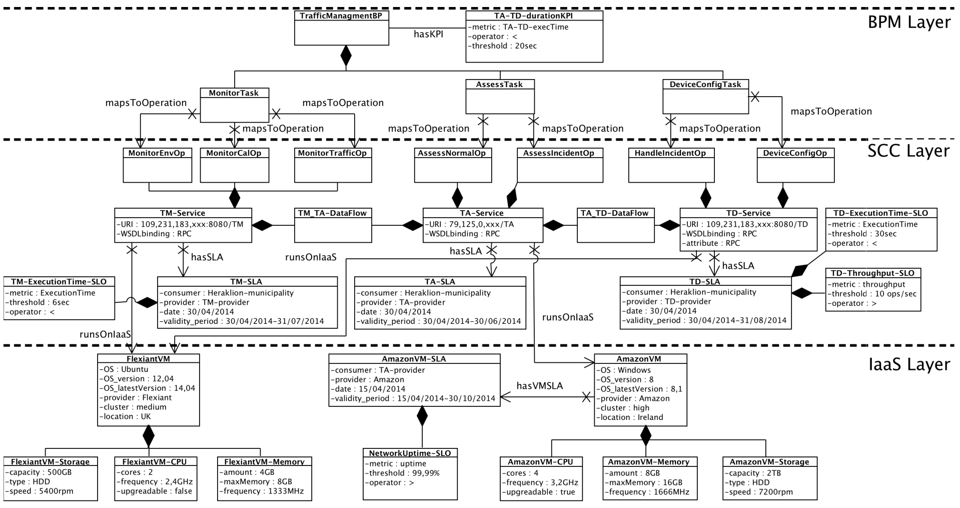


Figure 6.4: Sample component model of the traffic management application

Listing 6.7: Sample SQL query validating the dependency of two traffic management app's components

```
SELECT COUNT(*)
FROM (SELECT VM_COMPUTE_LIST
         FROM VM WHERE ComponentID=7001) as computeList
WHERE componentID IN (7026,7027)          AND (state = active);
```

6.3 Processing of Event Patterns

This section elaborates on the processing of the discovered patterns both on logical and temporal basis. The discovered patterns can directly be processed by mapping them to specific adaptation strategies mechanism, incorporated in the Adaptation Engine. However, in order to restrict the patterns considered by the adaptation engine, we apply a post-processing of the discovered patterns, aggregating and enhancing them using logical and temporal operators.

The Event meta-model presented in section 5.1 includes all the appropriate properties to express the logical and temporal relationships among the events comprising an event pattern. The former one is defined by the *logicalOp* attribute, while the latter one is defined by the *interval* attribute, dictating the maximum time interval between the occurrence of two consecutive events in the discovered event pattern. This means that if the second event of an event pattern does not appear within this time interval, then we disregard the considered pattern and either wait for other patterns with the same initial events, if there are any, or otherwise trigger corresponding adaptation rules for any critical events included in these considered initial events. For instance, we assume that the pattern discovery process has identified the following association rules: (i) $\{ce_1, we_2, we_4 \rightarrow E3\}$; and (ii) $\{ce_1, we_2, ce_5 \rightarrow E3\}$ with maximum waiting time for the occurrence of the we_4 and ce_5 events, 5 and 7 seconds respectively. If no event appears after 5 seconds of the e_2 occurrence, then we wait another 2 seconds to check if the second pattern is detected; otherwise, we trigger the adaptation rule corresponding to the critical ce_1 event (i.e., reactive adaptation).

ECMAF also provides a strict proactive mode that allows us to trigger an adaptation rule corresponding to the association rule with the higher λ score, after having received some initial events but there are two or more discovered patterns with different remaining events. Thus, we can be proactive both for the considered SLO violation, but also for any other warning or critical events in the triggered association rule. For instance, if we have the above two discovered association rules and assume that rule (ii) has higher λ score than this of rule (i), then in this strict mode, the system will trigger the adaptation rule corresponding to this rule, thus addressing the expected violation of $E3$ metric, but may also prevent critical event ce_5. This mode provides this twofold proactive functionality, however it involves the risk of triggering unnecessary or even unsuitable adaptation strategies, thus increasing the cost of the adaptation process.

As far as the logical interrelationships between the event patterns are concerned, we support the logical operators *AND*, *OR* and *NOT* for their post-processing. For instance, we consider the following discovered patterns:

(1) $\{ce_1, we_2, ce_4, we_6 \rightarrow E3\}$

(2) $\{ce_1, we_2, we_5, we_6 \rightarrow E3\}$

(3) $\{e_4, e_1, e_5 \rightarrow E3\}$

(4) $\{e_4, e_2, e_5 \rightarrow E3\}$

(5) $\{e_4, e_4, e_5 \rightarrow E3\}$

(6) $\{e_4, e_5, e_5 \rightarrow E3\}$

(7) $\{e_4, e_6, e_5 \rightarrow E3\}$

(8) $\{ce_1, we_2, ce_4, we_5, we_7 \rightarrow E3\}$

(9) $\{ce_1, we_2, we_5, ce_4, we_1 \rightarrow E3\}$

The post-processing of the above discovered patterns results in three composite association rules to be addressed by the Adaptation Engine, responsible for mapping them to suitable adaptation strategies. In particular, association rules (1) and (2) result in the composite association rule (a), i.e. either of the events ce_4 or we_5 may appear between event we_2 and we_6. Rules (3)–(7) are aggregated in the composite association rule (b) dictating that any event other than e_7 may appear in the second position of the pattern (we assume that events e_1–e_7 appear in the event stream and event e_3 is disregarded when searching for patterns leading to violations of the E_3 aggregate metric). In addition, rules (8) and (9) result in the composite association rule (c), combining the *AND* operator for events ce_4 and we_5 that may occur in either order and *OR* operator for we_7 and we_1 events.

(a) $\{ce_1, we_2, (ce_4 \ \vee \ we_5), we_6 \rightarrow E3\}$

(b) $\{e_4, \neg e_7, e_5 \rightarrow E3\}$

(c) $\{ce_1, we_2, (ce_4 \ \wedge \ we_5), (we_7 \ \vee \ we_1) \rightarrow E3\}$

So far, we have identified and processed domain specific event patterns, comprising events that are uniquely identified by their ID. For instance, a critical event of the metric e_1 indicates a violation of the available free memory of the VM hosting the Monitoring and Device Configuration services. Thus, this type of event patterns are domain specific and are valid only for the specific deployment of the considered SBA. However, in our approach, apart from the logical and temporal pattern aggregation described above, further processing of the patterns is performed, in order to identify domain independent patterns, thus leveraging the various contexts in which the SBA is performed. This can be achieved by aggregating the already discovered patterns, based on the type of metric (part of the event definition) and on the unique identifier of the raw events. For instance, two events indicating a violation of the CPU utilization of two different VMs can be aggregated to refer to the same abstract metric violation. In this way, we avoid long event patterns containing enough “noise”.

6.4 Conclusions

In a nutshell, this chapter analyzes the monitoring engine of the proposed ECMAF framework and the proposed event pattern discovery algorithm for identifying event patterns (association rules) leading to specific SLO violations. Particularly, the first part provides details about how cross-layer monitoring is performed, the formal definitions of metrics and monitoring events, as well as formal notations for the SLOs and the TSDB entries. The second part introduces a novel pattern discovery technique correlating monitoring events from various layers of a multi-Cloud SBA and thus enabling proactive adaptation and discovery of new component dependencies. The latter part also exploits instances of the component model through suitable SQL queries on the involved components. Finally, this chapter stands as a precursor for the next chapter dealing with the ECMAF’s Adaptation Engine.

Chapter 7

Adaptation

Contents

In the previous chapter we analyzed the functionality of the ECMAF's monitoring engine, as well as we introduced a novel pattern discovery technique for critical event patterns leading to specific SLO violations. The main goal of this chapter is to complete the description of the proposed framework, by presenting the functionality of the incorporated Adaptation Engine and thus completing the main part of this dissertation. Specifically, Section 7.1 provides an overview of the engine's components and the proactive and reactive adaptation capabilities of the ECMAF. Section 7.2 deals with the Adaptation Rule Manager performing the rule-based adaptation. In Section 7.3 we introduce the proposed technique for mapping the discovered event patterns to the most suitable adaptation strategies. In addition, this section addresses the configuration of the enacting adaptation actions and the generalization of the adaptation rules. Finally, Section 7.4 provides an adaptation scenario of the traffic management application to exemplify the functionality of the adaptation engine and Section 7.5 concludes this chapter.

7.1 Cross-layer Proactive and Reactive Adaptation

After having analyzed the functionality of the Monitoring Engine, in this section we continue the ECMAF's description with the Adaptation Engine, which provides both reactive and proactive adaptation capabilities. Figure 7.1 isolates the architecture of the adaptation engine to facilitate the reader for the rest of this chapter. Very briefly, this architecture comprises an *Adaptation Rule Manager*, which is responsible for mapping the discovered monitoring event patterns (passed from the Metrics Aggregator component) to suitable adaptation strategies, as well as for triggering the most suitable adaptation rules, through the incorporated rule engine. The adaptation strategy dictated by the selected rule is enacted by the corresponding *adaptation enactment mechanism*. The following two subsections elaborate on the proactive and reactive adaptation capabilities of ECMAF. Specific examples are given in Section 7.4, where our approach is validated on a traffic management case study.

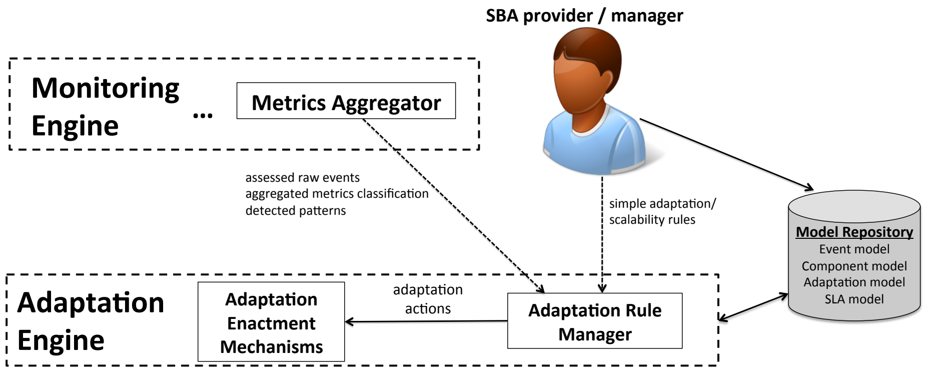


Figure 7.1: The ECMAF's adaptation engine

Proactive adaptation

It is very common that failures at one layer may lead to other failures and violations both to the same or to other layers, forming a chain of monitoring events. So, it is desirable to detect those event patterns causing service failures as early as possible. Our approach relies on using such a technique, presented in the previous chapter, to detect patterns of monitoring events, in order to prevent future critical SLO violations. As such, it enables the *proactive service adaptation* by mapping the detected patterns to suitable adaptation strategies, as analyzed in Section 7.3.

Specifically, the event patterns discovered by the Pattern Discoverer component are sent directly to the Metrics Aggregator component to detect them at runtime, while the Pattern Discoverer is also informed about the detected patterns, in order to omit them in the forthcoming executions of the pattern discovery algorithm. Upon pattern detection, the Metrics Aggregator urges the Adaptation Rule Manager to fire the respective adaptation rule (i.e. proactive adaptation) dictating an adaptation strategy's application, realized by the Adaptation Enactment mechanisms, before the actual SLO violation occurs.

Reactive adaptation

However, the ECMAF framework also reacts immediately on detected critical events, thus performing also reactive adaptation on the running SBA. Upon the detection of a SLO violation (i.e. a violation of a metric's critical threshold, defined in the WSLA document), the Adaptation Rule Manager immediately fires the corresponding adaptation rule, dictating the application of a specific adaptation action. Very briefly, as the rule derivation process is analyzed in Section 7.2, some predefined simple rules, mapping critical events to specific adaptation actions, are manually produced by the SBA Adaptation Manager (usually the SBA provider herself/himself) and are passed to the Adaptation Rule Manager. Then, these simple rules are used for mapping the discovered event patterns to suitable adaptation strategies.

As analyzed in the next chapter, the predefined simple adaptation rules are expressed in the DRL language (introduced in Section 3.4). For instance, going back to the traffic management running example, the Listing 7.1 defines an adaptation rule fired on an emergency pollution situation at the monitored area. This rule dictates a series of adaptation actions performed by the Device Configuration Service.

Adaptation enactment

Finally, as far as adaptation enactment is concerned, the Adaptation enactment mechanisms are responsible for realizing the adaptation strategy defined by the rule fired by the Adaptation Rule Engine. This rule also provides the appropriate input, defining which component is responsible for which adaptation action, the order of these actions, as well as their proper configuration. Such mechanisms have already been introduced in Section 4.2.1. The Infrastructure Manager component is able to treat some malfunctions regarding the IaaS layer, which is the main source of many SBA failures. For example, it can address adaptation actions, such as vertical and horizontal scaling, memory reallocation and others. Moreover, the Execution Engine, which will be designed as an extension of the existing BPEL engine with adaptation capabilities, is called to apply adaptation strategies regarding the SaaS layer (i.e., BPM and SCC layers).

7.2 Rule-based Adaptation

The adaptation rule derivation process is the most important part of the adaptation engine's functionality, as it defines the adaptation strategies to be enacted. Prior to this, the Adaptation Rule Manager requires the monitoring event patterns for the executed SBA, in order to determine suitable adaptation strategies, which may comprise one or more simple adaptation actions, as they are defined in the adaptation actions meta-model (see Section 5.2). Initially, the SBA Adaptation Manager maps single monitoring events to specific adaptation actions. Each

Listing 7.1: Sample adaptation rule in DRL for an emergency high pollution case of the traffic management example

```
rule "Emergency␣actions-␣Environment/Calendar/Traffic"
    when
       $action: Action(cruciality == "emergency␣action")
    then
                System.out.println("Actions␣Performed:␣\n");

                AlertSign alertSign = new AlertSign();
        String alertSignAction = alertSign.setSignalText("Very␣High␣
            pollution!␣-␣Please␣prefer␣using␣the␣public␣transport",
            $action.getArea(),"");

                TrafficLight trafficlight = new TrafficLight();
            String trafficLightAction =  trafficlight.setOperation("
                blinking␣orange", $action.getArea(),"");
            alertSignAction +=alertSign.setSignalText("Traffic␣lights␣are␣
                blinking␣orange!", $action.getArea(),"");

            Ring ring = new Ring();
        String ringAction = ring.setRingMode(true, $action.getArea());
        alertSignAction +=alertSign.setSignalText("Ring␣is␣enabled",
            $action.getArea(),"");

                TrafficPolice trafficPolice = new TrafficPolice();
        String trafficPoliceAction = trafficPolice.deviateTraffic("",
            $action.getArea());
        alertSign.setSignalText(trafficPoliceAction,$action.getArea(),"");
                alertSignAction +=trafficPoliceAction;

        Ambulance ambulance = new Ambulance();
        String ambulanceAction = ambulance.gotoIncidentLocation("",
            $action.getArea());

        String lowLevelActions = trafficLightAction  + alertSignAction +
            ringAction + ambulanceAction;
        $action.setLowLevelActions(lowLevelActions);
end
```

event can be mapped to more than one adaptation actions, but the SBA adaptation manager prioritizes them according to their appropriateness. The Adaptation Rule Manager extracts only one rule for each monitoring event (the one with the most suitable strategy), but we keep knowledge of the other adaptation strategies and their priority. Listing 7.2 contains sample adaptation rule of the traffic management application. For brevity reasons these rules are informally defined. However, as already discussed in Section 3.4, the adaptation rules of the running example are formally defined in the DRL language. In these rules, *R* stands for Rule and *As* represents their Appropriateness score. The rule with priority 1 is the most appropriate.

Listing 7.2: Sample adaptation rules of the traffic management example with their appropriateness scores

```
Flexiant application server failure -> server restore (R 7.1, As 1)
Flexiant application server failure -> server update (R 7.2, As 2)
Monitoring service execution time violation ->
       service substitution (R 7.3, As 1)
Monitoring service execution time violation ->
       server restart (R 7.4, As 2)
Asssessment service availability violation ->
       memory reallocation (R 7.5, As 1)
Assessment service availability violation ->
       service substitution (R 7.6, As 2)
Device Configuration service input mismatch ->
       service re-execution  (R 7.7, As 1)
Temperature sensor battery level below 5%(warning event) ->
       battery replacement (R 7.8, As 1)
```

Progressively, the Rule Manager creates rules that map patterns of monitoring events to the most suitable adaptation strategy/ies, aiming at addressing as many monitoring events as possible. We assume that each unique adaptation action has a unique ID, in order to facilitate the strategy matching. In order to achieve the best matching of adaptation strategies, we use the following technique, represented in Figures 7.2–7.6, based on the Cartesian product set theory. *S(e)* symbolizes the set of adaptation strategies assigned to the monitoring event *e*. We assume in all the cases above that the adaptation sets are examined sequentially, i.e. in the same order the corresponding events appear in the event pattern.

- **Case 1:** The optimum solution derives from the intersection of the adaptation strategies, i.e. $((S(e_1) \cap (S(e_2)) \cap S(e_3) \neq \emptyset$. If the intersection is not an empty set, then the event pattern is associated to all the strategies of the intersection, but only one of them is selected to be the most suitable one, according to strategies priority (Figure 7.2). In particular, the rationale is to multiply the priorities and take the intersection with the highest combined priority. When there are rules that concern two events independently, as well as one rule that combines them in a pattern, we would combine these two events with the third event, giving higher priority to the rule with the pattern which is greater than the multiplication of the priorities of the rules with the individual events. Small products mean high priority. This case does not make use of Cartesian products, as the results are retrieved from a single set (i.e. the intersected area).

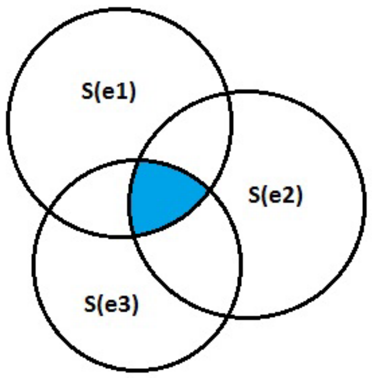


Figure 7.2: Case 1: Intersected area of the adaptation strategy sets

- **Case 2:** The worst solution is the situation, where there is no intersection among the strategy sets, i.e. $((S(e_1) \cap (S(e_2)) \cap S(e_3) = \emptyset$. In this case we choose the sets' union, i.e. the adaptation strategy with the highest priority for each one of the monitoring events (Figure 7.3). Moreover, the other strategy combinations are stored and assigned a priority, i.e. the multiplication of the individual strategies. In this case, the result S containing candidate adaptation actions is the Cartesian product of the considered disjoint sets:

$$S = (S(e_1) \times (S(e_2)) \times S(e_3) \tag{7.1}$$

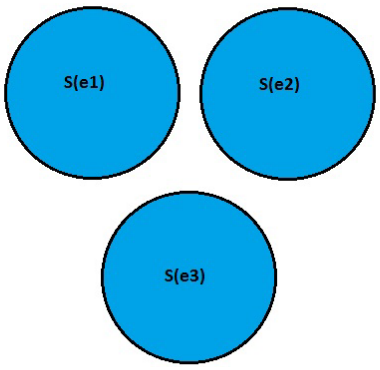


Figure 7.3: Case 2: Union of the strategy sets

- **Case 3:** Another intermediate case involves overlapping sets, whereas there is no disjoint set. For instance, if $S(e_1)$ is overlapped with $S(e_2)$ and $S(e_2)$ is overlapped with $S(e_3)$, then the resulted adaptation strategy will comprise two adaptation actions from the overlapped areas (Figure 7.4). In this case the result is the Cartesian product of the intersected areas:

$$S = (S(e_1) \cap (S(e_2)) \times (S(e_2) \cap (S(e_3)) \tag{7.2}$$

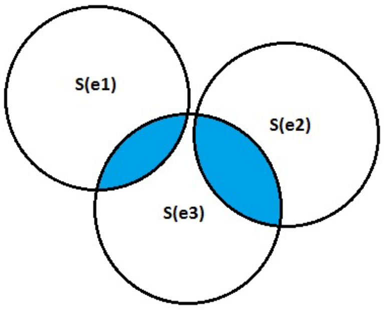


Figure 7.4: Case 3: Two overlapped strategy areas, no disjoint set

- **Case 4:** Another intermediate case involves both overlapping and disjoint sets. For instance, if $S(e_1)$ is overlapped with $S(e_2)$ and $S(e_3)$ does not overlap with any other set, then the resulted adaptation strategy will comprise two adaptation actions from the overlapped areas (Figure 7.5). The result in this case is the Cartesian product of the overlapped area and the disjoint set:

$$S = (S(e_1) \cap (S(e_2)) \times S(e_3) \tag{7.3}$$

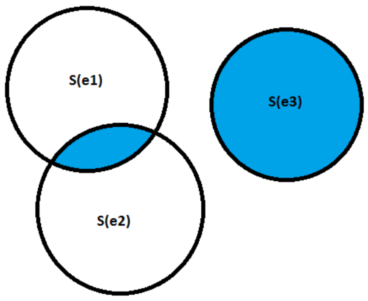


Figure 7.5: Case 4: Overlapped and disjoint strategy sets

- **Case 5:** Finally, another case involves many overlapping adaptation sets overlapping. For instance, we assume the following case: sets $S(e_1)$, $S(e_2)$ and $S(e_3)$ are intersected, sets $S(e_3)$, $S(e_4)$ and $S(e_5)$ are intersected too, while set $S(e_6)$ overlaps $S(e_2)$ and $S(e_4)$ (Figure 7.6). The resulted adaptation strategy will comprise three adaptation actions from the overlapped areas: (i) one adaptation action (a_1) (the one with the highest appropriateness score) from the intersection of $S(e_1)$, $S(e_2)$ and $S(e_3)$ sets, (ii) one adaptation action (a_2) , from the intersection of $S(e_3)$, $S(e_4)$ and $S(e_5)$ sets; and (iii) an adaptation action (a_3) from the intersection $S(e_2) \cap S(e_6)$, as this set contains an action with higher appropriateness score than the one in $S(e_4) \cap S(e_6)$. Therefore, the following rules apply when it comes to determine the most appropriate adaptation strategies:

1. If there are more than one intersected areas, we choose the one which involves the more individual adaptation strategy sets. Then, from this intersection we distinguish the adaptation strategy with the highest appropriateness score.

2. If there are two or more intersected areas involving the same number of individual sets, then we select the adaptation action with the highest appropriateness score from all the considered intersections.

3. The adaptation actions are executed sequentially in the order of their occurrence in the discovered event pattern.

Thus, the result in this case (Figure 7.6) is the Cartesian product of the three intersected areas:

$$S = ((S(e_1)\cap(S(e_2))\cap(S(e_3))\times((S(e_3)\cap(S(e_4))\cap(S(e_5))\times(S(e_2)\cap(S(e_6)) \tag{7.4}$$

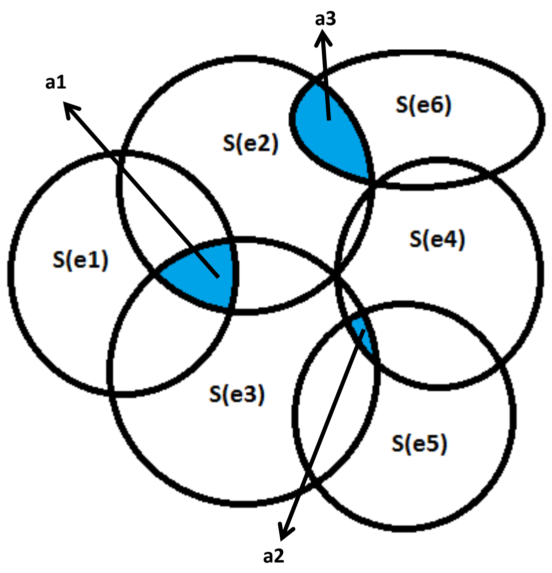


Figure 7.6: Case 5: Many intersected adaptation action sets

Going back to the traffic management example, the Adaptation Rule Manager would extract the following complex adaptation rules (Listing 7.3). These rules map the discovered event patterns to suitable adaptation strategies, based on the four cases described above and define the appropriateness score of each strategy.

Listing 7.3: Sample adaptation rules mapping event patterns to adaptation strategy/ies and their appropriateness

```
server failure + execution time violation + availability violation  ->
    server substitution (Case 1: derived from Rules 7.2, 7.4, 7.6) (R 7.9,
     As 8)
sensor battery drain + service input mismatch -> battery replacement +
    service re-execution (Case 2: derived from Rules 7.7, 7.8) (R 7.10, As
     1)
```

7.3 Mapping Event Patterns to Adaptation Actions

Until now, we have introduced a method for deriving suitable adaptation strategies for the discovered event patterns and calculating their appropriateness scores, based on the simple adaptation rule defined by the SBA Adaptation Manager. However, in order to calculate the final priority score *PS* for each adaptation rule, we also take into consideration the cost and the performance (i.e. the execution time) of the triggering adaptation actions.

As far as the cost is concerned, we discern two cases: (i) The adaptation action is performed on a public Cloud. Then, the cost exclusively depends on the cost model of the Cloud provider. For instance, the cost of a "scaling-up" action from a "medium" to a "high" VM varies among Cloud providers. (ii) The adaptation action is performed on a private Cloud. In this case, the cost depends on the execution time of the adaptation action and its resource utilization (i.e. the required amount of CPU, memory and storage).

In the literature many cost-optimization techniques have been proposed, aiming at minimizing the cost of enacting adaptation actions. In [Freitas et al. 2011] the authors propose Qu4DS framework, which aims at increasing the provider profit by dynamically managing services and resources, taking into account SLA prices, fines, and infrastructure costs. In our approach we are inspired by the work presented in [Leitner et al. 2013], which formalizes the problem of finding the optimal set of adaptations, taking into account the minimization of the total costs of SLA violations and the adaptations to prevent them. Three generic types of adaptation actions are considered: (i) Data manipulation actions, where some SBA data are changed, but not the service composition itself, (ii) Service rebinding actions, which are in fact service substitutions;and (iii) Structural adaptations, which alter not only the data flow of a service composition, but also the service composition itself. Each instance of these adaptation action types has a constant cost. For instance, the cost of substituting the Monitoring Service is actually the cost of using the new service minus the cost of using the old Monitoring Service. However, we additionally have to consider the penalty of the SLO violation (in the case of reactive adaptation), which has actually triggered the adaptation rule dictating the specific action. This cost is explicitly defined in the *Service Level* UML class of the Cloud component meta-model instance. In particular, the violation of each monitored property at each service level is mapped to a specific penalty. The higher the service level, the higher the imposed penalty.

We assume that a single adaptation action or a subset of the available adaptation actions (i.e. an adaptation strategy) denoted as $A = \{a_1, a_2, \ldots a_n\}$, is mapped to a SLO violation or to a discovered event pattern. Then, the total cost *TC* of the enacted adaptation strategy is the sum of costs of the incorporated enacted actions *C(A*)* ($A^* \in P(A)$, where P(A) is the powerset of A) and *P(m)* is the penalty dictated by the service level of the SLA document for the violation of the specific metric (Equation 7.5).

$$TC(A^*) = P(m) + C(A^*), A^* \in \mathcal{P}(A), \mathcal{P}(A) \to \mathbb{R} \tag{7.5}$$

Thus, in order to optimize the cost of the enacted adaptation actions, the service provider should minimize the total cost $TC(A^*)$. We assume in our approach that every adaptation rule refers to a single SLO violation, that is mapped to a constant penalty, defined by the SBA provider. Thus, the cost optimization process lies on the minimization of the total cost of the enacted adaptation actions dictated by the triggered adaptation rule (Equation 7.6).

$$C(A^*) = \sum_{a_k \in A^*} C(a_k) \to min \tag{7.6}$$

As far as the performance is concerned, it refers to the execution time and/or throughput of the adaptation action. We assume that performance is retrieved using specific benchmarks on the deployment VMs, for both private and public Clouds. These benchmarks are executed offline, before the disposal of the SBA to the final users. Thus, the SBA manager takes into account the performance, the cost and the appropriateness of all the supported adaptation actions for the specific application and decides on the most suitable ones upon the detection of SLO violations.

Finally, the appropriateness of the adaptation actions lies on the past experience of the SBA manager and mainly depends on the SBA deployment, i.e. if it is single- or multi-Cloud, which are the Cloud providers, the supported adaptation actions, the expected number of clients, the type of the involved Web services (internal or third-party) and generally the SBA context (type of SBA stakeholders, interrelationships with external services and devices, etc.). The appropriateness is measured in a [1-10] scale where 1 defines the most appropriate action, while 10 the action that most loosely addresses the detected SLO violation.

In order to have a reliable method for prioritizing the rules inserted by experts in the system, each one of the three criteria is assigned a suitable weight. We assume that the most critical criterion is the appropriateness of the adaptation action, then its cost and finally its performance. However, the SBA Adaptation Manager may personalize the weights of these criteria in a different way. Furthermore, as adaptation actions may fail (e.g. the system does not return to a consistent state or during the adaptation enactment another unexpected failure occurs), another factor is considered to lower the priority score when such failure actions appear. This penalty is given a higher weight in order to ensure that the failing adaptation action is not considered as the most suitable action and its priority score falls behind the score of the next adaptation rule. Equation 7.7 calculates the priority score *PS* of the simple expert rules, taking into consideration the above assumptions, where *a* represents the appropriateness of the action, *c* its cost, *p* its performance, *fa* the failing actions so far and *A* the total enactments of the adaptation actions, triggered by the involved rule.

$$PS = 0,3 * a + 0,2 * c + 0,1 * p - 0,4 * \frac{fa}{A} \tag{7.7}$$

The adopted weights of the three adaptation actions' priority criteria dictates that the priority *PS* is measured in [0,06 - 0,6] scale, as all criteria are normalized in a [0,1-1] scale. We do not normalize in a [0-1] scale, because the rule with the lowest priority will always take zero appropriateness value. The normalization is performed on actions that are fired by rules with the same head. For instance, if we have three rules mapping the same event pattern to different adaptation actions (Listing 7.4), then appropriateness for adaptation action a_1 (Equation 7.8)is normalized using the following linear function, where *NA* is the normalized appropriateness, *a* the appropriateness value, *max(a)* is the maximum defined appropriateness and *min(a)* is the minimum defined appropriateness.

Listing 7.4: Mapping an event pattern to different adaptation actions and their appropriateness

```
a+b+c -> Action a1 (R 7.11, As = 10)
a+b+c -> Action a2 (R 7.12, As = 8)
a+b+c -> Action a3 (R 7.13, As = 3)
```

$$NA(a_1) = 0,1 + \frac{(a - min(a)) * (1 - 0,1)}{max(a) - min(a)} \tag{7.8}$$

Using this formula, the appropriateness of the second adaptation action a_2 (Equation 7.9) is calculated as follows:

$$NA(a_2) = 0,1 + \frac{(8 - 3) * (1 - 0,1)}{10 - 3} \simeq 0,67 \tag{7.9}$$

The same formula is used to normalize the cost and performance values. The priority score of the adaptation strategies, comprising two or more actions, is calculated by multiplying the priority scores of the individual adaptation actions. Consequently, this final score $P(A^*)$ of the selected enacted adaptation strategy again falls in the range [0,1-1] ($P(A^*) \in [0.1 - 1]$).

After the calculation of the normalized scores *NA* of the available adaptation actions using the formula 7.8, we can perform an overall ranking. This ranking is dynamic, as some adaptation actions may be ranked lower if they frequently fail. Thus, the adaptation manager will choose the highest ranked adaptation action each time.

7.3.1 Adaptation Actions Configuration

The adaptation mechanisms, enacting the actions dictated by the triggered adaptation rules, require that the corresponding actions are sufficiently and promptly configured with all the appropriate parameters, so as to have the best possible result (reactive or proactive). The adaptation actions meta-model, presented in Section 5.3, provides the required properties for the adaptation actions' configuration. The XML or JSON files describing the enacting adaptation actions include these configuration properties, which are extracted by the file parser. For instance, going back to our running example, the XML file in Listing 7.5 describes the adaptation action's configuration of the triggered adaptation rule handling the detection of pattern $p_1 = \{ce_1 \rightarrow we_2 \rightarrow we_4\}$ (see Listing 8.13).

Listing 7.5: Sample XML adaptation action file of the traffic management example

```xml
<?xml version="1.0"?>
<Adaptation_strategy>
   <Simple_adaptation_action actionID=4321>
      <name>IaaS_Vertical_scaling</name>
      <performer>OpenStack adaptation engine</performer>
      <cost>10</cost>
          <fired_on>1407747500</fired_on>
          <affected_components>MonitoringService,DeviceConfigService</
              affected_components>
      <manual_activity>false</manual_activity>
      <cloud_provider>OpenStack</cloud_provider>
      <location>Greece</location>
      <CPU>4-cores</CPU>
      <memory>6GB</memory>
      <disk>750GB</disk>
      <new_resource_class>medium</new_resource_class>
   </Simple_adaptation_action>
</Adaptation_strategy>
```

7.3.2 Rule Generalization

The technique used for generalizing and aggregating the discovered event patterns (see Section 6.3) can also be exploited for generalizing the adaptation rules. This generalization is performed by the adaptation engine and it is useful for the Adaptation Manager to keep rules well-ordered and helps her/him easily delegate events and/or event patterns to the corresponding adaptation mechanisms. However, in this case, the head of the rules are aggregated based only on logical disjunctions, as they involve only single entities (events or patterns). For instance, we assume that we have the following rules mapping simple events and discovered event patterns to adaptation actions.

(1) $\{ce_1, we_2, ce_4, we_6 \rightarrow a_1\}$

(2) $\{ce_1, we_2, we_5, we_6 \rightarrow a_1\}$

(3) $\{e_4, e_1, e_5 \rightarrow a_2\}$

(4) $\{e_4, e_2, e_5 \rightarrow a_2\}$

(5) $\{e_4, e_4, e_5 \rightarrow a_2\}$

(6) $\{e_4, e_5, e_5 \rightarrow a_2\}$

(7) $\{e_4, e_6, e_5 \rightarrow a_2\}$

(8) $\{ce_1, we_2, ce_4, we_5, we_7 \rightarrow a_3\}$

(9) $\{ce_1, we_2, we_5, ce_4, we_1 \rightarrow a_3\}$

(10) $\{e_4 \rightarrow a_4\}$

(11) $\{e_2 \rightarrow a_4\}$

The above adaptation rules will result in four composite rules. In particular, rules (1) and (2) result in rule (a), i.e. either of the events ce_4 or we_5 may appear between event we_2 and we_6 to trigger action a_1. Rules (3)–(7) are aggregated in rule (b) dictating that any event, except form e_7 may appear in the second position of the pattern to trigger action a_2, while rules (8) and (9) result in the composite association rule (c), combining the *AND* operator for events ce_4 and we_5 that may occur in either order and *OR* operator for we_7 and we_1 events, to trigger action a_3. Finally, composite rule (d) aggregates simple rules (10) and (11) involving only simple events.

(a) $\{ce_1, we_2, (ce_4 \lor we_5), we_6 \rightarrow a_1\}$

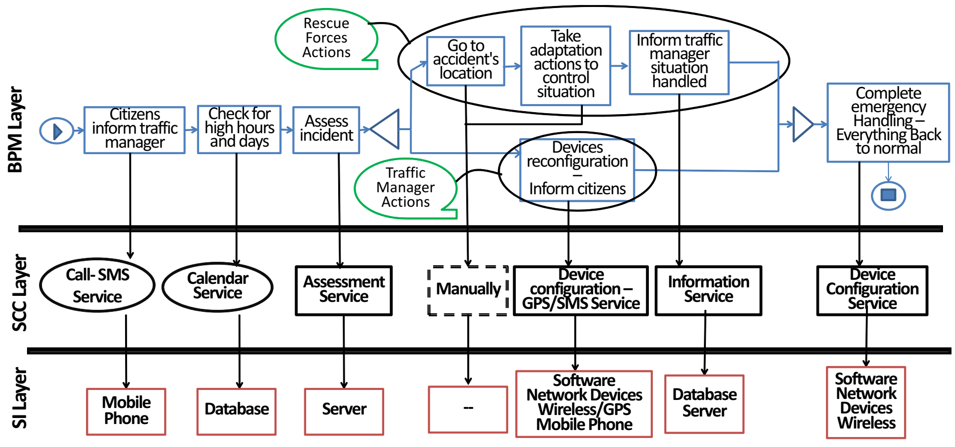

BPM Layer
SCC Layer
SI Layer
Rescue Forces Actions
Go to accident's location
Take adaptation actions to control situation
Inform traffic manager situation handled
Citizens inform traffic manager
Check for high hours and days
Assess incident
Complete emergency Handling – Everything Back to normal
Traffic Manager Actions
Devices reconfiguration – Inform citizens
Call-SMS Service
Calendar Service
Assessment Service
Manually
Device configuration – GPS/SMS Service
Information Service
Device Configuration Service
Mobile Phone
Database
Server
–
Software Network Devices Wireless/GPS Mobile Phone
Database Server
Software Network Devices Wireless

Figure 7.7: Critical traffic conditions - Traffic management scenario

(b) $\{e_4, \neg e_7, e_5 \rightarrow a_2\}$

(c) $\{ce_1, we_2, (ce_4 \ \wedge \ we_5), (we_7 \ \vee \ we_1) \rightarrow a_3\}$

(d) $\{e_4 \ \vee \ e_2 \rightarrow a_4\}$

7.4 Running Example

In this section, we elaborate on a traffic management scenario to exemplify the reactive and proactive adaptation capabilities of the proposed framework. Figure 7.7 describes the critical traffic conditions process of our case study. The business process workflow with the involved activities is presented at the BPM layer. At the SCC layer, the Web services realizing these activities are offered. The oval shapes represent the third party services, the rectangles the internal services, while the dashed ones the manual activities. Finally, at the SI layer, the underlying infrastructure for each service of the SBA is supported.

In the following example we show how our approach can be used to proactively adapt this traffic management system. Initially, we suppose that the domain expert specifies in a separate document his goals for the quality of the traffic management system using a set of KPIs, directly related to the monitored properties. In addition, she/he defines adaptation actions, as part of simple adaptation rules, that can be performed in order to improve the business process, such as possible activity parallelism and cost optimization techniques. As far as the involved Web services are concerned, the corresponding SLAs for both third party and internal services are stored in the model repository. Usually SLAs define failing events, while some warning events are defined for information purposes. Thus, it is possible that the whole set of warning events is not covered by a single SLA and the additional warning events must be covered manually. In our case the Adaptation Manager is interested in and has subscribed for all the events regarding the traffic management system. Furthermore, the following manual simple adaptation rules (Listing 7.6) have been imported in the Rule Engine's production memory. Both rules concern warning events.

Listing 7.6: Adaptation rules of the traffic management application

```
(execution time of assessment service > 10s) ->
       memory reallocation (R 7.14, As 1)
 (execution time of assessment service > 15s) ->
       service substitution (R 7.15, As 1)
(execution time of assessment service > 10s) ->
       service re-execution (R 7.16, As 2)
(available memory < 100MB) -> memory reallocation (R 7.17, As 1)
```

At run time, we continually collect monitoring events using the monitoring tools. Successful events, which are the majority of all the events, indicate that the system is running normally and the adaptation manager does not have to take any actions. After a certain period of time, we notice that the Assessment Service is not running optimally and the monitoring tool detects warning events indicating that its raw execution time has exceeded the warning threshold of 10, but not the critical threshold of 15. Almost at the same time, the IaaS monitoring tool detects that the server hosting the Assessment service, which decides on the traffic actions to be taken after an accident, does not run properly because of low available memory. In particular the available memory is below 100MB, which is the warning limit. Both events are passed directly to the Monitor Manager and delivers them to the Adaptation Engine through the incorporated publish/subscribe system. However, this event sequence has appeared many times during past SBA instance invocations and the Pattern Discoverer has derived a corresponding event pattern (response time warning, low available memory). Then, the Adaptation Rule Manager, following the method analyzed in Section 7.2, maps this pattern to the most suitable adaptation strategy, according to the predefined manual adaptation rules (Listing 7.7).

Listing 7.7: A complex adaptation rule of the traffic management application

```
(execution time of assessment service > 10s) + (available memory < 100MB)
    -> memory reallocation + service re-execution (derived from Rules
    7.16, 7.17) (R 7.18, As 1)
```

Rule 7.18 derives from rules 7.16 and 7.17 and has priority 1, that is the multiplication of the corresponding priorities. The *memory reallocation* adaptation strategy stems from the intersection of the adaptation strategies of the two warning events. As soon as the Metrics Aggregator detects this pattern, the rule 7.18 is fired, dictating that the best adaptation strategy is to reallocate memory in the server hosting the assessment service, as well as to re-execute the assessment service. The suitability of this strategy lies on the fact, that by re-executing this service with better memory allocation, the probability that its response is not violated becomes very high (as we do not know if another failure may occur in the near future regarding the new instance) and in this way, also a KPI violation may be avoided. The derived strategy involves IaaS layer actions. Consequently, the Adaptation Manager provides the appropriate configuration to the responsible adaptation enactment mechanisms. After the memory reallocation, the Assessment service's execution time is kept in normal ranges, below 250ms, and there is now enough available memory. Thus, it is obvious that this technique provides cross-layer adaptation capabilities to the ECMAF framework, as it may extract adaptation strategies involving various actions, spread across the SOA and Cloud layers of a SBA.

In the previous example, it is illustrated that the proposed framework efficiently handles such a cross-layer scenario. Proactive adaptation is achieved by exploiting the pattern of two warning monitoring events to prevent a future failing event (SLA violation). Moreover, the dependencies between the layers are clearly discerned and the performed adaptation action addresses the detected malfunctions. However, the ECMAF framework immediately reacts (i.e. reactive adaptation) upon the detection of critical events. For instance, if the execution time of the Assessment Service exceeds the critical threshold of 15s, then the Adaptation Rule Manager will immediately fire Rule 7.15, dictating a “service substitution” adaptation action.

7.5 Conclusions

To sum up, this chapter analyzes the functionality of the ECMAF’s adaptation engine. In particular, it is split into four main parts. The first one elaborates on the proactive and reactive adaptation capabilities of the engine, while the second one provides details about the functionality of the incorporated rule engine. The third part introduces the mapping method from event patterns to suitable adaptation actions, based on the appropriateness, the cost and the performance of the available suitable actions, as well as an adaptation rule generalization method. Finally, the fourth part provides a complete adaptation scenario of the traffic management example to exemplify the cross-layer reactive and proactive capabilities of ECMAF. This chapter concludes the analysis of ECMAF before going into implementation details in the next chapter.

Chapter 8

Implementation

Contents

After having analyzed the contributions of this dissertation, this chapter provides implementation details about the ECMAF framework and the traffic management application. Especially, we elaborate on the tools we have exploited and the coordination of the underlying techniques and languages introduced in section 3.4 to accomplish the functionality of the individual framework's components. The rest of the chapter is structured as follows: Section 8.1 provides details on the implementation of the multi-Cloud traffic management SBA, used as a running example throughout the dissertation. Section 8.2 studies the proposed meta-models realization and exploitation by the ECMAF's monitoring and adaptation engines, the implementation of which is analyzed in Section 8.3 along with illustrative examples of the traffic management application. Finally, Section 8.4 concludes this chapter.

8.1 Traffic Management Application's Implementation

In Chapter 2 we have thoroughly analyzed the workflow of the Traffic Management application and the functionality of its individual components. This section provides details about the Business Process and the individual Web services realizing the Traffic Management Business process activities.

For usability purposes, the incorporated monitoring mechanism [Konsolaki 2012] allows for the design of Business Process Model and Notation (BPMN 2.0)[1] models rather than abstract Business Process Execution Language (BPEL) [OASIS-BPEL 2007] processes. On the one hand, BPMN [White 2004] was developed by the Business Process Management Initiative (BPMI)[2] in 2005, in an effort to provide a notation that is readily understandable by all business users, ranging from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and, finally, to the business people who will manage and monitor those processes.

On the other hand, BPEL was developed in 2002 by a consortium consisting of BEA Systems (acquired by Oracle[3] in 2008), IBM[4] and Microsoft[5]. The BPEL specification defines the syntax and semantics of the BPEL language, which contains a variety of process flow constructs. It allows for conditional branching, parallel process flows, nested sub-processes, process joins and other related features. It distinguishes between two different kinds of business process: (i) executable processes and (ii) abstract processes. Whereas executable processes must contain all the details that are necessary to be executed by a BPEL engine (e.g. ActiveVOS[6], Apache ODE[7], etc.), abstract processes are not executable and can have unspeci-

[1]http://www.bpmn.org
[2]http://www.bpmi.org/
[3]http://www.oracle.com
[4]http"//www.ibm.com
[5]http://www.microsoft.com
[6]http://www.activevos.com/
[7]http://ode.apache.org/

fied or hidden parts.

As the ASTRO[8] monitoring tool (see details in the next section) that is utilized for monitoring the BPM and SCC layers, requires a BPEL executable file, it is desirable to map the BPMN diagram to an executable BPEL process. This mapping is a difficult endeavor trying to bridge the gap between business process design and implementation. This intrinsically complex mapping lies to the fact that there is a structural disparity between BPMN and BPEL. BPEL is a block structured language, while BPMN is a constrained, but relative free form graph. There are no fundamental difficulties in mapping a BPEL process to an isomorphic BPMN diagram, but the opposite mapping (BPMN to isomorphic BPEL) is not always possible. However, there is a number of business process tools performing and validating such conversions.

As far as the design of the traffic management business process (Figure 8.1) is concerned, we are exploiting ADONIS Business Process Management Toolkit[9] (Community edition), developed by BOC group[10]. ADONIS is based on the BPMS (Business Process Management System) framework (Figure 8.2) for process management and the establishment of a continuous process improvement cycle. This framework consists of four key elements: (i) products/services, (ii) processes, (iii) organizational structures; and (iv) information technology (IT). The goal of business process management is the optimization of business processes and corporate structures, as well as resources and technologies. This is achieved through a holistic approach to collect, evaluate, optimize and control the business processes. All these functions are provided by ADONIS tool.

The designed BPMN model is transformed to the corresponding abstract BPEL process using the ADONIS modeling tool. Then, this BPEL process is modified according to the BPEL 1.0 format required by the Astro monitoring tool and the general monitoring framework presented in [Konsolaki 2012] and is adopted by

[8]http://astroproject.org/

[9]http://www.adonis-community.com/

[10]http://www.boc-group.com/

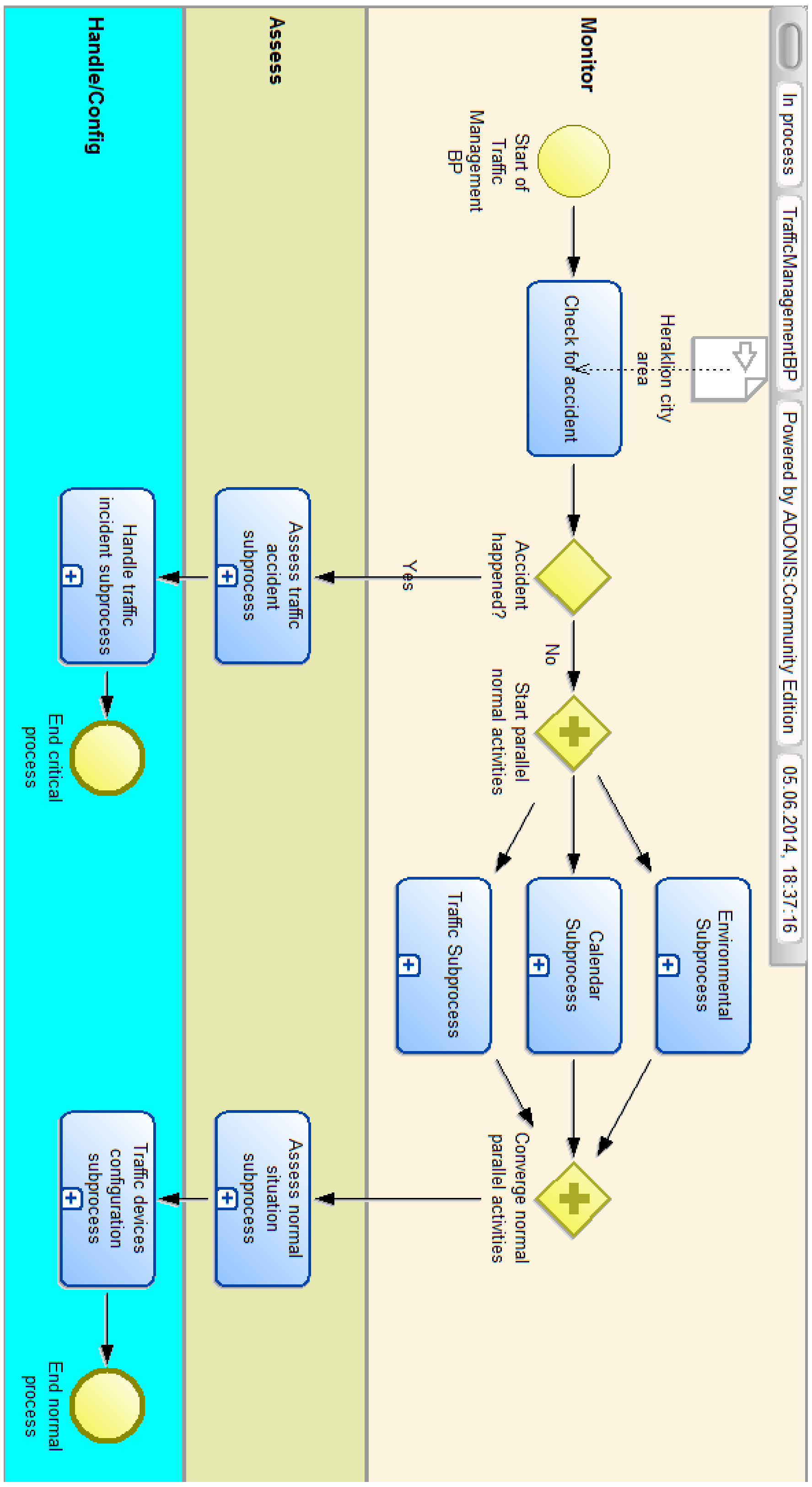

In process
TrafficManagementBP
Powered by ADONIS:Community Edition
05.06.2014, 18:37:16
Monitor
Assess
Handle/Config
Start of Traffic Management BP
Heraklion city area
Check for accident
Accident happened?
Yes
No
Start parallel normal activities
Environmental Subprocess
Calendar Subprocess
Traffic Subprocess
Converge normal parallel activities
Assess traffic accident subprocess
Handle traffic incident subprocess
End critical process
Assess normal situation subprocess
Traffic devices configuration subprocess
End normal process

Figure 8.1: The Traffic Management BPMN model designed in ADONIS tool

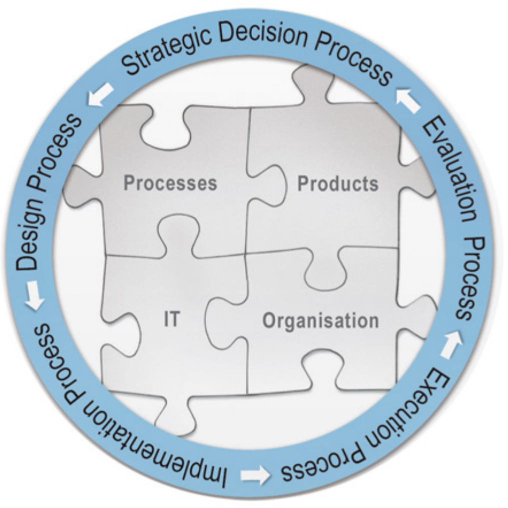


Figure 8.2: BOC ADONIS Process Management Framework

our approach.

After having available the abstract BPEL file, we map the abstract activities to specific Web services, as well as we provide the corresponding Web Services Description Language[11] (WSDL) file. WSDL is a XML-based interface description language that is used for describing the functionality offered by a Web service, and the choreography file, which contains the composition's partners, as well as the definition of the monitored properties.

As far as the three Web services comprising the Traffic Management SBA (i.e. the Monitoring service, the Assessment Service and the Device Configuration Service) are concerned, they have been implemented in Eclipse[12] integrated development environment (IDE). To achieve this we follow the procedure below:

1. We create a Web Service Endpoint Interface for each one of the three Web services using the RPC binding style (`@SOAPBinding(style = Style.RPC)`) required by the Astro monitoring tool. *RPC* style indicates that the SOAP message body contains an XML representation of a method call and uses the names of the method and its parameters to generate XML structures

[11]http://www.w3.org/TR/wsdl
[12]http://www.eclipse.org/

that represent a method's call stack, while the *Document* style indicates that the SOAP body contains a XML document, which can be validated against a pre-defined XML schema document. In this interface we define the Web service's operations as `webmethods`, identifying their input and return types. Listing 8.1 declares the *getTemperature* operation of the Monitoring Service.

Listing 8.1: The declaration of the getTemperature operation of the Monitoring Service

```
@WebMethod
String getTemperatureMeasurement(String location) throws IOException;
```

2. We create a Web Service Endpoint Implementation, implementing the corresponding interfaces. For instance, the definition of the Monitoring Service is expressed in the following Listing 8.2:

Listing 8.2: The declaration of the Monitoring Service implementation class

```
@WebService(endpointInterface = "monitor.ws.Monitor")
public class MonitorImpl implements Monitor { ... }
```

3. The implemented Web service is exported as a Web application ARchive (WAR) file from Eclipse IDE, containing all the appropriate libraries utilized by the Web service. This WAR file is located in the *webapps* folder of an Apache Tomcat[13] application service instance and it is deployed upon the application server's start-up. Then, the WSDL file of the deployed web service should be accessible via a URL. For example, the Monitoring Service's WSDL deployed on Flexiant VM Tomcat application server is available at the following URL (Listing 8.3):

Listing 8.3: The URL of the Monitoring Service WSDL file

```
http://109.231.122.186:8080/MonitorTask/monitor?wsdl
```

4. We implement a Web service client pointing to the WSDL URI to access the deployed Web service. For instance, to access the Monitoring Service, we

[13]http://tomcat.apache.org/

call it with the appropriate input (e.g. "area1" of the Heraklion city) and it provides a list of environmental, traffic and calendar measurements (Listing 8.4):

Listing 8.4: Environmental measurements detected by the Monitoring Service

```
-  30143#Traffic management app#Monitor Task#Temperature
    #1386611567937#23.0##SI#Environmental_Sensor
-  90893#Traffic management app#Monitor Task#NO2_level
    #1386611567940#50.0##SI#Environmental_Sensor
-  50185#Traffic management app#Monitor Task#SO2_level
    #1386611567940#20.0##SI#Environmental_Sensor
```

5. We define the choreography file of the business process utilizing the abstract BPEL file and the WSDL files using the Enterprise Architect[14] software.

8.2 Meta-models' Realization and Exploitation

This section deals with the design and exploitation of the meta-models presented in chapter 5. We have already argued in that chapter for the adoption of the UML modeling language to express these models. However, as UML models are not directly exploitable, we discuss how they can be transformed to be machine readable and be incorporated in the ECMAF framework.

We have used Visual Paradigm[15] tool for designing the three basic UML meta-models. This software allows for designing both UML class diagrams (e.g. Figure 5.4 at p. 100), as well as object diagrams, representing a static design of a system from a prototypical perspective (i.e., instances of the core meta-models (e.g. Figure 5.5 at p. 103). To the best of our knowledge, there is no automatic tool converting XML schemas to corresponding JSON schemas; thus, we have manu-

[14]http://www.sparxsystems.com/products/ea/
[15]http://www.visual-paradigm.com/

ally defined the latter ones using a useful JSON schema tool[16]. The usefulness of the dynamic instances of the proposed meta-models lies on the following facts: (i) The XML schema of the proposed event meta-model is utilized by the Metrics Aggregator component of the Monitor Manager to validate the detected monitoring events / measurements (i.e. to check if all the required information is provided), (ii) the component meta-model instance is exploited to validate the extracted event patterns, i.e. whether the source components are correlated with each other. This component model is dynamically adjusted whenever a system's component is affected by an adaptation actions or the pattern discovery process detect new component dependencies; and (iii) the adaptation actions meta-model instance is useful for the Adaptation Manager who is responsible for identifying the simple adaptation rules mapping simple monitoring events to suitable adaptation actions.

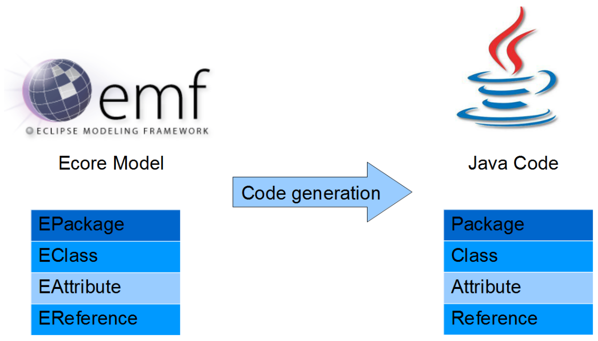


Figure 8.3: EMF code generation to Java classes

Especially, regarding the component meta-model, we are exploiting the model-driven technologies offered by Eclipse Modeling Framework (EMF)[17]. The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From the model specification described in XML Meta-data Interchange (XMI), EMF provides tools and run-time support to produce a set of Java classes for the considered model (Fig-

[16]http://www.jsonschema.net/
[17]https://www.eclipse.org/modeling/emf/

ure 8.3), along with a set of adapter classes that enable viewing and command-based editing of the model and a basic editor. The core meta-model is exported from the Visual Paradigm as a XML schema file, which, in turn, is imported in EMF to extract the base domain code. Then, we exploit Connected Data Objects (CDO)[18], which offer as a development-time model repository and a run-time persistence framework. CDO offers transactions with save points, explicit locking, change notification, queries, temporality, branching, merging, offline and fail-over modes. The storage back-end is pluggable and migrations between direct JDBC[19], Hibernate[20], Objectivity/DB[21], MongoDB[22] are provided. Figure 8.4 depicts the basic architecture of CDO.

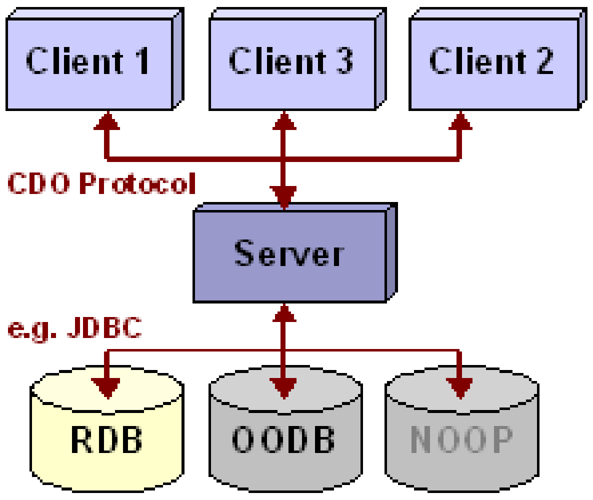


Figure 8.4: The CDO architecture

In our case, a MySQL database lies under a CDO server to store the component model instances. For the creation and configuration of this database we have used the phpMyAdmin[23] interface integrated with the MAMP[24] web server. This database is further exploited by the proposed pattern discovery algorithm

[18]http://wiki.eclipse.org/CDO
[19]http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[20]http://hibernate.org/
[21]http://www.objectivity.com/products/objectivitydb/
[22]http://www.mongodb.org/
[23]http://www.phpmyadmin.net/
[24]http://www.mamp.info/en/

to validate component dependencies through specific SQL queries. The query in Listing 8.5 returns the total number of active components from the FlexiantVM's (componentID=7001) compute components list, having ID 7026 or 7027 (i.e., the ID of the components producing events e_1 and e_2), in order to identify if both components reside in the same VM and thus affect each other when violations occur. However, the instance model of the CDO server needs to be modified when adaptation actions altering the state of one or more components occur or when new component dependencies are discovered from the pattern discovery algorithm.

Listing 8.5: Sample SQL query validating the components' dependencies

```
SELECT COUNT(*)
FROM (SELECT VM_COMPUTE_LIST
         FROM VM WHERE ComponentID=7001) as computeList
WHERE componentID IN (7026,7027)         AND (state = active);
```

8.3 ECMAF's Implementation

This section elaborates on the implementation of the ECMAF framework. In particular, Sections 8.3.1 and 8.3.2 provides implementation details for the Monitoring and Adaptation engines' components respectively.

8.3.1 Monitoring Engine

This section analyzes the implementation tools incorporated in the ECMAF framework to realize the Monitoring Engine. Figure 8.5 provides an overview of these tools analyzed in the next two subsections; the first one focuses on the individual monitoring components detecting the monitoring events, while the second one addresses the realization of the Monitor Manager.

Monitoring Components

In this section we focus on the implementation of the individual monitoring components distributed across the functional layers of a multi-Cloud SBA, in or-

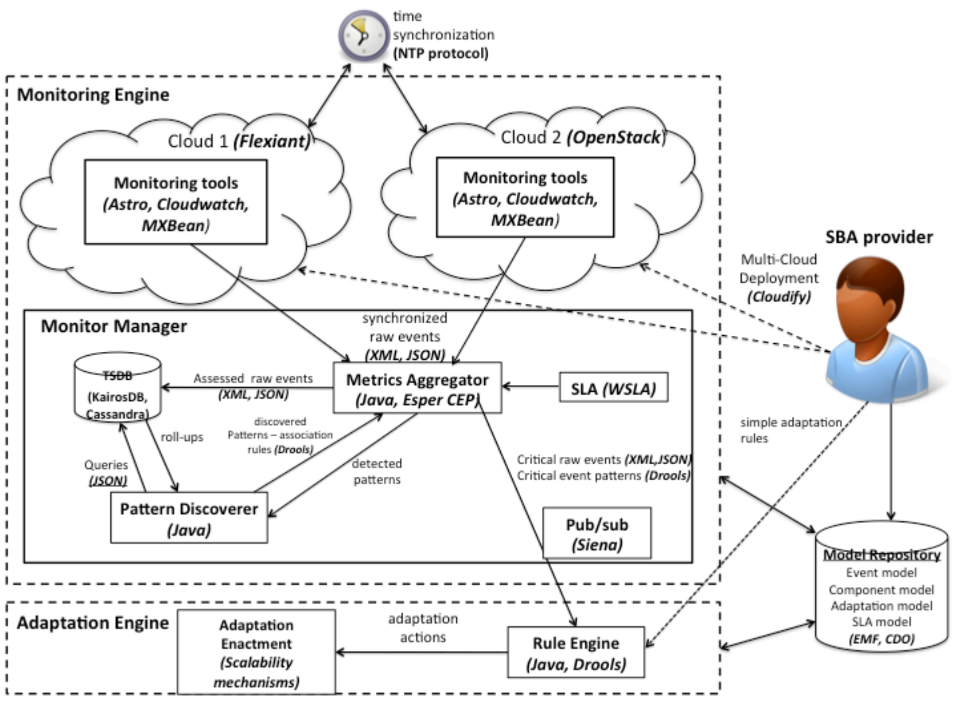

time synchronization (NTP protocol)
Monitoring Engine
Cloud 1 (Flexiant)
Monitoring tools (Astro, Cloudwatch, MXBean)
Cloud 2 (OpenStack)
Monitoring tools (Astro, Cloudwatch, MXBean)
SBA provider
Multi-Cloud Deployment (Cloudify)
Monitor Manager
synchronized raw events (XML, JSON)
TSDB (KairosDB, Cassandra)
Assessed raw events (XML, JSON)
Metrics Aggregator (Java, Esper CEP)
SLA (WSLA)
roll-ups
discovered Patterns – association rules (Drools)
Queries (JSON)
detected patterns
Critical raw events (XML,JSON)
Critical event patterns (Drools)
simple adaptation rules
Pattern Discoverer (Java)
Pub/sub (Siena)
Model Repository
Event model
Component model
Adaptation model
SLA model
(EMF, CDO)
Adaptation Engine
Adaptation Enactment (Scalability mechanisms)
adaptation actions
Rule Engine (Java, Drools)

Figure 8.5: ECMAF implementation overview

der to capture violations in both the SOA and Cloud layers stack. As already analyzed in Section 6.1.1, we exploit OWL-Q language for defining QoS metrics. These metric definitions mainly determine the frequency and the formulas for calculating the complex metrics based on the raw metrics provided by the following monitoring components.

As far as the SaaS layer monitoring is concerned, we use the Astro monitoring tool [Barbon et al. 2006]. Astro supports monitoring of service compositions implemented in Business Process Execution language for Web Services (BPEL4WS)[25] for both instance and class properties. Once the monitored properties are defined in OWL-Q, we transform them to a Real Time Markup Language (RTML) expression used by Astro. The supported metrics (QoS attributes) include the execution time (min, max), throughput (min, max, average) and availability of a Web service or SBA [Konsolaki 2012].

[25]http://xml.coverpages.org/bpel4ws.html

Astro requires as input a choreography file, containing the composition partners (i.e. the abstract BPEL processes and the corresponding WSDL documents) and the definition of the monitoring properties. The framework detects violations at the SCC and the BPM layers by comparing the monitoring property values against the metrics' thresholds defined in the corresponding WSLA document and stores them as a new entry to the TSDB. This entry contains the following attributes: (i) the event ID, (ii) the metric, (iii) the detected value, (iv) the event's timestamp, (v) the considered application, (vi) the considered application's task, (vii) the host ID, (viii) the layer, (ix) the component's ID (i.e. the component producing the event); and (x) the component's name.

The PaaS monitoring component exploits an existing cross-PaaS application management solution [Zeginis et al. 2013b] introduced in the Cloud4SOA EU project. This solution offers a Cloud technology-agnostic PaaS monitoring functionality (in addition to a SLA management layer) unifying diverse resource-level metrics provided by the individual Cloud providers. Supported metrics include application load, application and DB response time, application container response time and Cloud response time.

The IaaS monitoring component distinguishes between direct monitoring of the infrastructure and monitoring services offered by Cloud providers. For direct monitoring we implement the *OperatingSystemMXBean* interface[26] to get the following resource metrics: (i) VirtualMemorySize, (ii) PhysicalMemorySize, (iii) SwapSpaceSize, (iv) ProcessCpuLoad, (v) ProcessCpuTime, (vi) SystemCpuLoad, (vii) TotalPhysicalMemorySize; and (viii) TotalSwapSpaceSize. In addition, we exploit an auto-configuring wireless sensor network [Garefalakis and Magoutis 2012] based on the IEEE 802.15.4 (Zigbee)[27] protocol to collect real environmental metrics (e.g. temperature). Sensor data are collected and modeled as monitored entities giving us the ability to detect events from the SBA context.

[26] http://docs.oracle.com/javase/7/docs
[27] http://www.zigbee.org/

As an instance of Cloud monitoring services we use Amazon Cloudwatch[28], a Web service that provides comprehensive monitoring for Cloud resources and the applications that customers run on Amazon Web Services (AWS). To gain system-wide visibility of running EC2 VMs, we enable a variety of metrics through the Cloudwatch API, including CPU utilization, disk read/write rate and the volume of incoming/outgoing network traffic. Each Cloudwatch API request returns a datapoint that is stored in the TSDB and handled as a monitored entity. Our requests are issued every few seconds to ensure that the collected data are valid and we can still react to them at a reasonable latency.

Time Synchronization

One of the main goals of our approach is to identify particular event patterns occurring during SBA execution that lead to critical violations, so as to enable the selection of the appropriate cross-layer adaptation actions. Since the order of publishing events is significant, the monitoring events must be time-synchronized before being sent to the Adaptation Engine. Time synchronization is particularly important in multi-Cloud settings as standard time synchronization solutions are rarely deployed across Cloud providers.

Over time, a computer's clock is prone to drift. This is problematic as many network services require the all computers on a network share the same accurate time. The Network Time Protocol (NTP)[29] is one way to provide clock accuracy in a network, even if virtual machines are involved [VMware 2011], as in the multi-Cloud deployment of the traffic management application. The NTP protocol is specially designed to synchronize the time on a network of machines. It uses the concept of a stratum to describe how many NTP hops away is a machine from an authoritative time source, in order to properly synchronize it. Figure 8.6 illustrates the separation of an NTP network of machines in stratums (i.e. levels), indicating the distance of the reference clocks (stratum 0) depicted at the top of

[28]http://aws.amazon.com/cloudwatch/

[29]`http://www.ntp.org/`

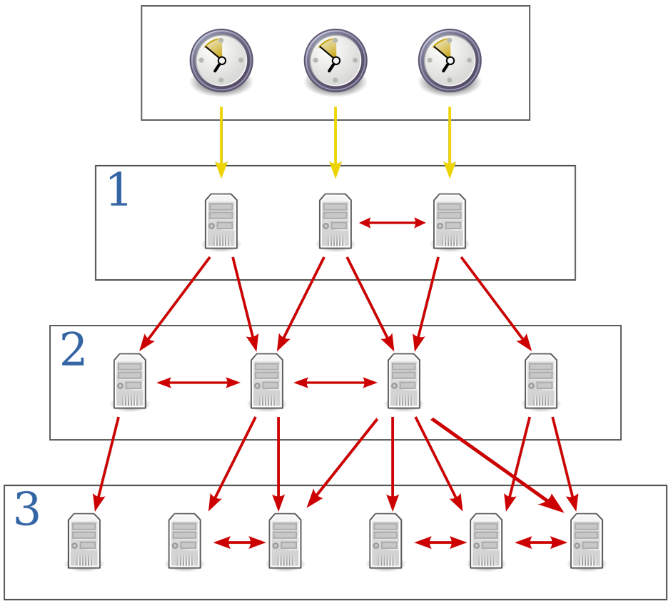


Figure 8.6: Time synchronization through the NTP protocol

the figure. Yellow arrows indicate a direct connection, while red arrows indicate a network connection. A configuration file determines which NTP server to query and the drift file stores the time drift for each machine. In our traffic management running example, the incorporated machines (i.e. Flexiant VM, OpenStack VM, local machine hosting the Monitor Manager and the Adaptation Manager) are timely synchronized using primarily the *"0.europe.pool.ntp.org server"* as the reference clock and all the underlying machines are at stratum 1 (Figure 8.7). Other alternative NTP clocks are also included in the configuration file.

Other clock synchronization algorithms, dealing with the temporal ordering of events, produced by concurrent processes, include: (i) those that use logical clocks to create event sequence numbers; and (ii) those that use physical clocks to synchronize events. Lamport's algorithm [Lamport 1978] makes use of physical clocks, i.e. it assigns unique timestamps (numbers) to all the events of the system to achieve a total or partial ordering of the events. In general, it requires a monotonically increasing software counter for a clock, that has to be incremented for each new event. Physical clock-based algorithms, such as Cristian's

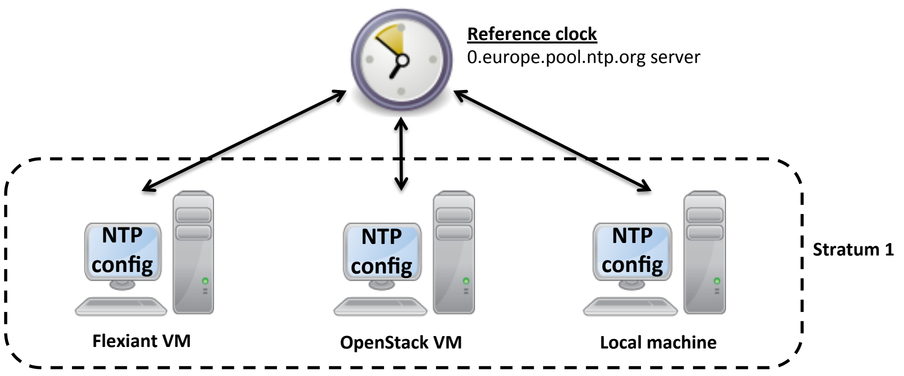


Figure 8.7: NTP time synchronization for the traffic management application VMs

algorithm [Cristian 1989] or Berkeley algorithm [Gusella and Zatti 1986] adjust the system clocks of the system components, based on server time or a master machine time accordingly.

Monitor Manager

SLO assessment As already briefly analyzed in Section 6.1.3, the SLO assessment is performed by the Metrics Aggregator component. This is a Java-based component that parses the WSLA document defining the SLOs and assesses the measured events against the SLO thresholds. Simple metrics (e.g. row execution time of the MonitorService, availability of the DeviceConfig service, etc.) are assessed upon their delivery, whereas composite metrics (e.g. average CPU_load of the flexiantVM, max throughput of the Assessment Service, etc.) are assessed according to the frequency dictated by the OWL-Q metric's definition. If the assessment indicates a warning or critical violation of the specified metric, the "criticality" field is filled accordingly and the assessed event is stored into the TSDB.

Pattern Discovery The pattern discovery algorithm introduced and analyzed in section 6.2 has been implemented in Java, utilizing two DSLs for processing the monitoring events stream and defining the association rules between the causing and caused events. The first one is Esper (see Section 3.4) that is used to process the monitoring events stream, collected by the TSDB storing the raw monitoring events, through a suitable JSON query (Listing 8.9). The second one is Drools Rule Engine (see Section 3.4), which is exploited to define the association rules (Listing 8.6), i.e., the rules defining the causal relationship between the event pattern and the caused SLO violation.

Listing 8.6: Sample association rule in DRL for the event pattern with ID=2001

```
rule "rule1-TM-TD-durationKPI"
    when
        (FlexiantVM-FreeMemory < 300MB) && (FlexiantVM-FreeMemory > 100MB) && (FlexiantVM_CPU_load
             > 95%)
        && (NetworkUptime < 98%) && (ExecTimeDeviceConfigService > 20sec) && (
             ExecTimeDeviceConfigService < 22sec)
    then
               violation(TM-TD-durationKPI)
end
```

As far as the event processing is concerned, the event stream retrieved from the TSDB via a JSON query is divided into groups of events containing the events (stored in a Hashmap with an incremental key (i.e. the event ID) and value a LinkedHashSet storing the time-ordered events) within the time interval defined by the considered aggregate metric (see Figure 6.3). Then, each Hashmap is processed to determine if it corresponds to a SLO violation or not. In the former case, the subset of events until the last critical event of the considered metric is examined, while in the latter one, all the LinkedHashSet's objects are processed (a separate Hashmap is used to preserve the interval number and the corresponding position of the last critical event of the considered metric (0 for non-critical intervals)). For the calculation of the power set of the individual LinkedHashsets, required by the Pattern Discovery Algorithm, we use the *powerset* function provided by the *google guava*[30] library. A limit of 30 events per interval is posed by this function. Although, in practice, it is unusual to have more than 30 warning or critical events within a an interval, an exception will be thrown if the metric definition requires infrequent measurement.

[30]https://code.google.com/p/guava-libraries/

Pattern Detection Another functionality provided by the Metrics Aggregator component is the event pattern detection. The discovered patterns identified by the Pattern Discoverer are sent to the Metrics Aggregator component, which is responsible for detecting them at run-time, in order to prevent the actual expected violation form happening, thus enabling proactive adaptation. For this reason, we are exploiting the Esper CEP engine, providing special features for pattern detection. EPL language can be used to define event patterns, similarly to the listeners defined in Listing 3.2 filtering incoming events. For instance, Listing 8.7 defines the EPL statement of the pattern with ID=2001 in our running example. This statement defines the order of events within the pattern and the max time between the first and the last event (e.g. 1 minute and 20 seconds).

Listing 8.7: EPL statement for pattern with ID=2001

```
SELECT * FROM pattern [every (FlexiantVM-FreeMemory(criticality=warning) -> FlexiantVM-CPU-load(
    criticality=critical)  -> NetworkUptime(criticality=critical) -> ExecTimeDeviceConfigService(
    criticality=warning)).win:time(1 minute 20 seconds)]
```

Similarly, we can define more complex EPL statements for patterns combining logical and time operations. For instance, we assume that the Pattern Discoverer identifies an event pattern comprising an e_1 event followed by an e_4 event within 20sec, but in the meantime there should be no e_6 event. Listing 8.8 formally defines this pattern, as well as the composite patterns introduced in section 6.3L.

Listing 8.8: EPL statements for complex event patterns

```
SELECT * FROM pattern [every ($e_1$ → not $e_6$ → $e_4$).win.length(20)]

SELECT * FROM pattern [every ($ce_1$ → $we_2$ → $we_4$.win.length(5))]

SELECT * FROM pattern [every ($ce_1$ → $we_2$ → $ce_5$.win.length(7))]

SELECT * FROM pattern [every ($ce_1$ → $we_2$ → ($ce_4$ and $we_5$) → $we_6$)]

SELECT * FROM pattern [every ($e_4$ → not $e_7$ → $e_5$)]

SELECT * FROM pattern [every ($ce_1$ → $we_2$ →($ce_4$ and $we_5$) → ($we_7$ or $we_1$))]
```

Storing Events KairosDB, presented in Section 3.4, can store data points of any type of metric. When pushing data in the database we need to provide the unique

ID of the metric. The KairosDB required fields for the inserted data points are the following: Metric name, Value, Timestamp, other tags to distinguish between the different measurements (e.g., VM id, cloud provider, VM-type, application name, task name). Additionally, KairosDB by default provides a series of internal metrics to monitor the application's performance. These metrics are written to the data store one per minute and their statistics can be viewed on the KairosDB Web interface (Figure 8.8). However, one can adjust how often they are reported by suitably modifying the kairosdb.properties file. These internal metrics are the following:

- **JVM statistics:** Free memory, Max memory, Threads
- **API statistics:** HTTP requests statistics, Telnet statistics
- **Datastore statistics:** Query time, Write size (i.e., the number of rows written to the data store during the last write)

KairosDB comes with a series of aggregators that perform an operation over all data points of a specific metric that exist in the sampling period. Aggregators can also be combined together and are processed in the order they are specified in the imported JSON query file. The output of one aggregator is sent as input to the next. For example, the JSON query in Listing 8.9 includes two aggregators on the jvm.total.memory metric. The first one divides data points of the specific time interval by 1024 and the second one calculates the average value of the divided data point values.

All aggregators, except rate and div, allow downsampling, i.e., reducing the sampling rate of the data points and aggregating these values over a long period of time. The basic KairosDB aggregators that the user can easily modify/combine to provide added value functionality are the following:

- **avg** - returns the average value
- **dev** - returns the standard deviation

Listing 8.9: sample JSON query on the KairosDB including two aggregators

```
{
  "metrics": [
    {
      "tags": {},
      "name": "kairosdb.jvm.total_memory",
      "aggregators": [
        {
          "name": "div",
          "divisor": "1024"
        },
        {
          "name": "avg",
          "align_sampling": true,
          "sampling": {
            "value": "1",
            "unit": "milliseconds"
          }
        }
      ]
    }
  ],
  "cache_time": 0,
  "start_absolute": 1391464800000,
  "end_absolute": 1391551200000
}
```

- **div** - returns each data point divided by a divisor
- **histogram** - calculates a probability distribution and returns the specified percentile for the distribution
- **least_squares** - returns two points for the range which represent the best fit line through the set of points.
- **max** - returns the largest value
- **min** - returns the smallest value
- **rate** - returns the rate of change between a pair of data points.

- **sum** - returns the sum of all values

KairosDB provides two methods for pushing data (i.e. the raw monitoring events in our case). The first and the simplest one is via telnet. The syntax is concise and easy to use. For instance, the following command in Listing 8.10 is used for pushing data:

Listing 8.10: Pushing the monitoring events in the KairosDB via Telnet

```
put <metric name> <timestamp> <value> <tag> <tag>... /n
```

The second method is the REST API, which can be exploited through HTTP requests for adding and/or querying measurement data for metrics. If one wants to batch large amounts of data, she/he can gzip a JSON file and upload it with the content type set to "application/gzip". The general structure of the JSON file is illustrated in Listing 8.11. The first entry inserts three data points of one metric, while the second one adds a single data point of another metric.

Figure 8.8 depicts the KairosDB graphical Web interface, which provides useful functionalities, such as aggregation of multiple metrics in specific time ranges, using the incorporated aggregators mentioned below, queries in JSON and JS Object format and metric data points plotting. These functionalities (except the plotting one) are provided by the KairosDB API, which has been extended to meet our needs for more comprehensive JSON support, so as to be easily processed by the Pattern Discoverer component, which requires an event stream as a JSON file, containing an array of data points.

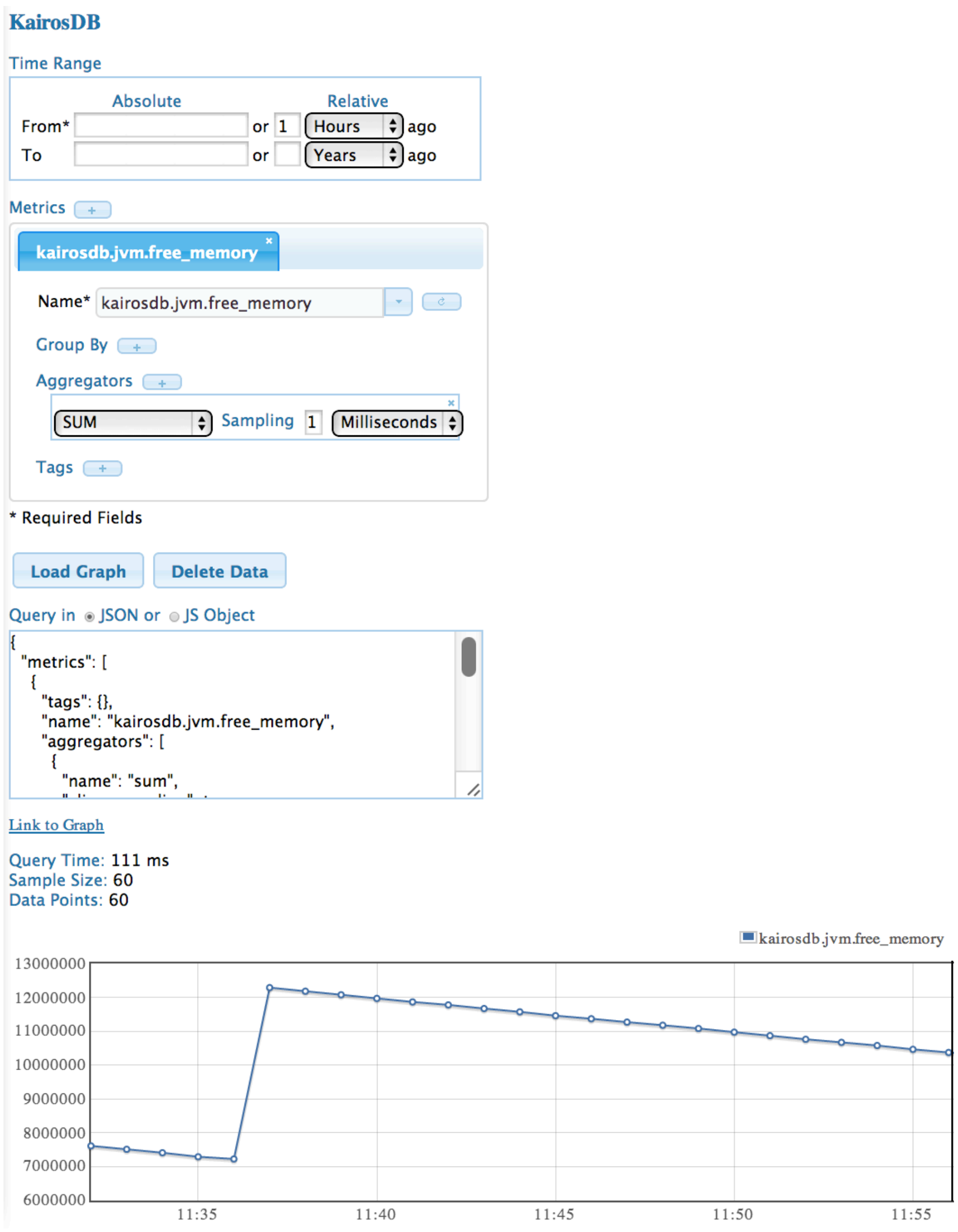
Publishing / Subscribing Events Concerning the publish / subscribe system, utilized by ECMAF to filter the SLO violations and the detected event patterns leading to SLO violations transferred to the adaptation engine, we use the Scalable Internet Event Notification Architectures (Siena)[31] publish/subscribe event notification service. Siena is expressive enough to capture all appropriate event information through an extensible data model without sacrificing scalability and

[31]http://www.inf.usi.ch/carzaniga/siena/

Listing 8.11: Pushing the monitoring events in the KairosDB via the REST API

```
[
    {""
        name:"CPU-"usage,""
        datapoints:[[ 1391464800, 87%],[1391464920, 73%],[1391465040, 92%]
            ],""
        tags:{""
            Cloud_provider:""Flexiant,"
            VM-"ID:"7001"
            ...
        }
    },
    {""
        name:"Memory-"usage,""
        timestamp: 1391464800,""
        value:469,""
        tags:{""
             Cloud_provider:""AmazonVM,"
             VM-"ID:"7033"
                         ...
        }
    }
]
```

performance during event delivery. The technical basis of Siena is an innovative type of network service called content-based networking [Carzaniga et al. 2011], which is a novel communication infrastructure in which the flow of messages through the network is driven by the content of the messages, rather than by explicit addresses assigned by senders and attached to the messages. Through this publish / subscribe mechanism, the Adaptation Manager subscribes only to specific metrics, for which she/he is interested. Thus, whenever a metric's violation (i.e. reactive adaptation) occurs or a critical event pattern (i.e. proactive adaptation) leading to a violation of the subscribed metrics is detected, the Metrics Aggregator immediately sends them to the Adaptation Engine to efficiently handle the malfunction, exploiting the incorporated rule-based adaptation mechanism.



From* or 1 Hours ago
To or Years ago

Name* kairosdb.jvm.free_memory

SUM Sampling 1 Milliseconds

* Required Fields

Query in [✓ JSON] or JS Object

```
{
  "metrics": [
    {
      "tags": {},
      "name": "kairosdb.jvm.free_memory",
      "aggregators": [
        {
          "name": "sum",
```

Query Time: 111 ms
Sample Size: 60
Data Points: 60

Figure 8.8: The KairosDB web interface

8.3.2 Adaptation Engine

Adaptation Rules

Regarding the adaptation rules, they can be efficiently modeled in the proposed event meta-model through the incorporated relevant classes (see Section 5.1).

Thus, the *Event* part of the ECA triplet is described by the Event class, the *Condition* is described by the warning and critical SLOs of the corresponding metric, the Action is expressed through the *Action* class, while the whole rule is uniquely identified by the *Adaptation_Rule* class.

In the ECMAF framework, the adaptation rules are realized using the Drools Rule Language (DRL)[32] presented in Section 3.4. DRL is exploited to define both simple adaptation rules mapping critical events to adaptation actions (Listing 8.12), as well as more complex rules mapping event patterns to suitable adaptation strategies (Listing 8.13), as they are determined by the corresponding technique, analyzed in Section 7.3.

Listing 8.12: A Simple DRL adaptation rule of the traffic management application

```
rule "LowMemoryScaling␣Rule"
    when
        AvailableMemory(OpenStackVM) < 100MB
    then
        scale_up("high␣VM") //Provided that the current deployment is on a
             "medium" VM
end
```

Listing 8.13: An adaptation rule mapping an event pattern to a scaling action in DRL

```
rule "Pattern1␣Rule"
    when
        DetectedPattern(Pattern1) // Pattern p1 = ce1 -> we2 -> we_4
    then
        scale_up(medium VM) AND substitute(AssessService,newAssessService)
             //Provided that the current deployment is on "low" VM and
            there is an equivalently functional Assessment Service
end
```

[32]http://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch05.html

Adaptation Enactment Mechanisms

As far as the *Adaptation Enactment* is concerned, it can be realized by various tools and depends on the type of the adaptation actions. It is mainly part of ongoing and future work to implement as many adaptation sub-engines as possible, that can realize the adaptation actions defined in the proposed adaptation actions meta-model (see Section 5.3). However, some basic adaptation actions are already realized, especially the ones mapping to the mechanisms provided by the Cloud providers (e.g. horizontal and vertical scaling). For instance, if an adaptation action dictates a "scaling up" action of a deployment instance, then this action may be performed by the scaling mechanism of the Cloud provider. Other adaptation actions may require human intervention (e.g. replacement of a sensor used by a service or start up of a local machine).

Regarding the scaling actions provided by the Cloud providers, in this section more details are provided for the OpenStack[33] Cloud computing platform, which is utilized by the traffic management SBA, as the primary Cloud Computing platform. OpenStack is primarily deployed as an IaaS solution and it can be used for building both private and public Clouds. The technology consists of a series of interrelated projects that control pools of processing, storage and networking resources throughout a data center, able to be managed or provisioned through a Web-based dashboard, command-line tools or a RESTful API.

Figure 8.9 depicts the basic architecture of an OpenStack VM. In order to perform the actions dictated by the triggered adaptation rules, we employ a basic OpenStack setup consisting of:

1. A **controller node**, which provides the necessary services for managing the VM. The controller node contains one network interface in the management network. The experimental setup employs: a Database (Oracle MySQL RDBMS), a Message broker (Apache Qpid with enabled), a synchronization service (NTP) and virtualization services (VMware for controller and net-

[33]http://www.openstack.org/

work nodes and Kernel-based VM (KVM) for the compute nodes). Other basic services include: (i) Identity (Keystone) service, which is responsible for the authentication and authorization of the interaction and communication between a user and a service, (ii) Image (Glance) service, which enables users to discover, register and retrieve VM images. Its functionality is augmented by using a REST API that enables users to query virtual machine image meta-data and retrieve an actual image, (iii) Compute (Nova) service, which is a Cloud computing fabric controller which compasses all sorts of components from computing processes to networking modules, (iv) Networking (Neutron) service, which is responsible for all the networking infrastructure among various OpenStack services; and (v) Dashboard (Horizon) that is the all-in-one management user interface for configuring OpenStack. This allows users to deploy/destroy instances, observe resource usage of VM instances, apply security policies etc.

2. A **network node**, which is responsible for hosting the services that provide networking functionality to compute instances. In particular, they host the DHCP server (Quantum DHCP Agent), meta-data proxy services and Virtual Bridging services (Open-vSwitch + Quantum Agent) with tunneling and Virtual Routing (Quantum L3 Agent) services. The network node contains one network interface in the management network, one in the instance tunnels network and one in the external network.

3. At least one **compute node** deployed on demand of the adaptation engine. Compute nodes form the resource core of the OpenStack Compute cloud, providing the processing, memory, network and storage resources to run instances. The compute node contains one network interface on the management network and one on the instance tunnels network.

OpenStack uses by default five type of VMs (Figure 8.10), based on the storage and computation power, called flavors. However, the user may edit his/her own

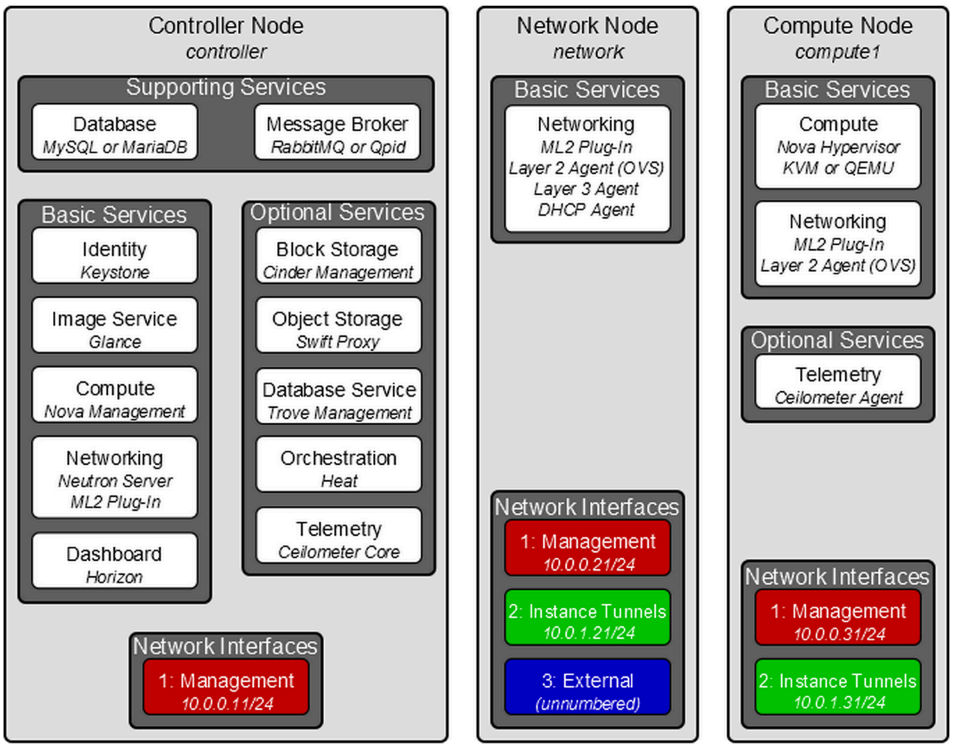
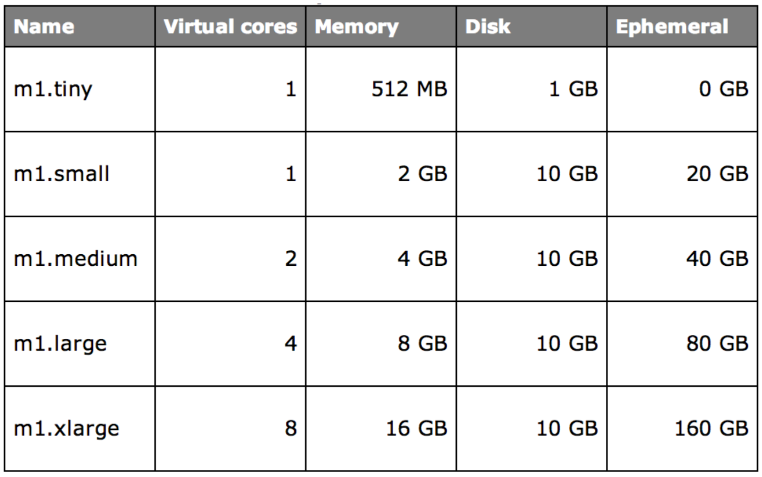


Figure 8.9: The three-node architecture of OpenStack

templates, choosing the virtual cores, the memory and the disk of the provisioned VM on demand, based on the configuration of the enacted adaptation action.



Name	Virtual cores	Memory	Disk	Ephemeral
m1.tiny	1	512 MB	1 GB	0 GB
m1.small	1	2 GB	10 GB	20 GB
m1.medium	2	4 GB	10 GB	40 GB
m1.large	4	8 GB	10 GB	80 GB
m1.xlarge	8	16 GB	10 GB	160 GB

Figure 8.10: The default flavors of OpenStack

8.4 Conclusions

To sum up, in this chapter we have analyzed how we have implemented and tested the proposed ECMAF framework. The first part addresses issues regarding the implementation of the Traffic Management SBA, used throughout the dissertation as a running example. The second part deals with the realization of the proposed meta-models and focuses more on the component model, that is exploited by the pattern discovery algorithm, while the last part discusses the implementation of the monitoring and the adaptation engines.

Chapter 9

Evaluation

Contents

This chapter is devoted to a thorough evaluation of the ECMAF framework. Three sets of experiments are conducted and presented in the following sections. The vast majority of the experiments focuses on evaluating performance scalability, in terms of computation time, by varying specific parameters. The first set examines the ECMAF's monitoring engine and attempts to investigate the effect of all parameters that are of interest in each case. The second set of experiments evaluates the adaptation engine, while the third one evaluates the overall evaluation of the complete framework.

For the purposes of the evaluation, we use the traffic management application deployed on two VMs, according to their individual requirements (Section 1.4).

Apart from the events detected by the deployed monitoring components, we also generate synthetic events in order to have an adequately large sample dataset for the pattern discovery and detection subsystems, as well as for the mapping detrimental event patterns to suitable adaptation actions. The experiments, except the adaptation ones, are performed on a machine with quad-core CPU 2.6GhZ, 8GB RAM memory and Mac OS X Mavericks operating system. The computation time values are an average of 20 runs.

9.1 Monitoring Evaluation

This section focuses on the evaluation of the ECMAF's monitoring engine. More specifically, Section 9.1.1 evaluates the performance of the TSDB, while Section 9.1.2 evaluates the performance and accuracy of the Pattern Discoverer component.

9.1.1 TSDB Evaluation

In this section we describe an experimental evaluation of our TSDB storage system, for monitoring data under three different deployments: (i) running on a single node, (ii) three nodes running in the same Cloud provider; and (iii) three nodes running under different Cloud providers (multi-Cloud). In addition, we calculate the end-to-end latency under the first setup, to have a clear view of our monitoring engine's total response time.

In particular, our storage system consists of a KairosDB using Cassandra as a database store. To distribute the traffic between the origin servers (in the multiple node setups), we run a load balancer at the top, which is responsible for delivering the HTTP requests (queries) at the TSDB nodes equally. As a result, we distribute the workload among the nodes and we add additional redundancy to our setup when a server fails.

To demonstrate the performance and scalability of our storage system we collect one million events, also known as data points, using the monitoring components and we store them into the TSDB. Then, these events are passed to the Adaptation Engine through the Siena publish/subscribe mechanism.

To evaluate the system, we retrieve different numbers of data points from the total amount of one million and we measure the response time of the system as well as the total time (i.e. end-to-end latency) to pass the events from the TSDB to the Adaptation Engine. We run the same queries in three different setups, as shown in Figure 9.1. In the first setup we have the TSDB running on a single virtual machine. Afterwards we deploy TSDB and the other components in three different virtual machines running on the same cloud provider (Flexiant). In the last setup we have three virtual machines running in different cloud providers, Amazon, Azure and Flexiant. The total time refers to single node setup.

Table 9.1 summarizes the response time results of the approaches above. For the queries of five thousand and ten thousand data points, the three approaches are really close in performance level. If we request larger sets of data the single node approach is getting really slow, making it a bad choice in a heavy workload system. On the other hand, the other two approaches behave equivalently, so we believe that the slight increase in latency at the multi-cloud approach is caused by the cross network communication between different Cloud providers. Since the events we want to monitor, appear across multiple Clouds, choosing the last setup would give us data locality and the advantage to store the events in the same cloud they occur, avoiding additional network overhead. As far as the total time is concerned, the publish/subscribe mechanism exhibits significant performance, as it can efficiently pass huge amounts of events to the adaptation engine within reasonable latencies.

Table 9.1: KairosDB response time under different setups / end-to-end latency

Data Points	**5k**	**10k**	**50k**	**100k**	**200k**	**500k**	**1M**
1 Node	136ms	180ms	475ms	752ms	1337ms	2436ms	4440ms
3 Nodes-Single Cloud	129ms	160ms	293ms	492ms	820ms	1988ms	2930ms
3 Nodes-Multi Cloud	130ms	165ms	302ms	505ms	842ms	2042ms	3112ms
end-to-end latency	1070ms	1970ms	8674ms	16302ms	35683ms	82526ms	161545ms

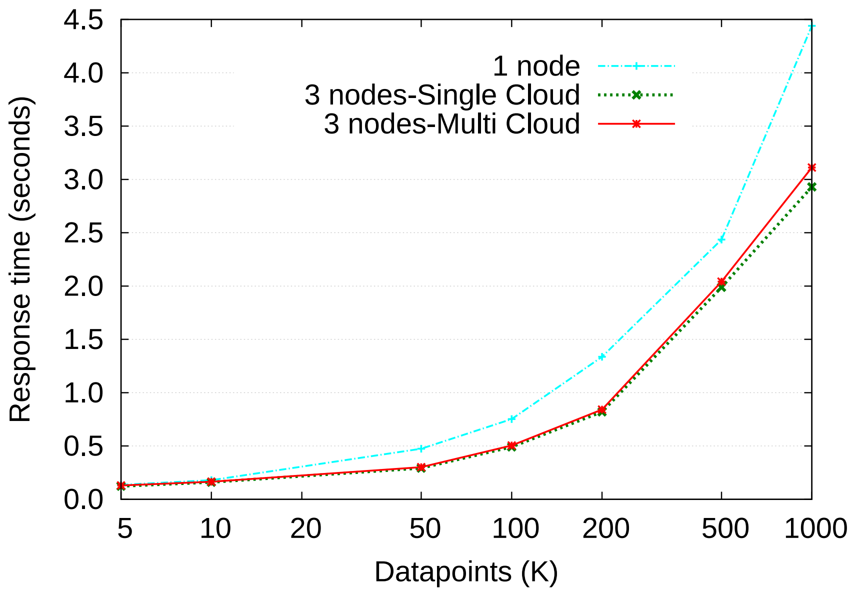


Figure 9.1: TSDB - Evaluation, Response time in read queries under different setups.

Table 9.2: End-to-end (TSDB+Siena) response time, throughput under different setups

Number of events published (K)	**5**	**10**	**50**	**100**	**200**	**500**	**1000**
Single query latency (sec)	0.59	0.82	1.5	2.21	3.68	7.65	11.88
Single query throughput (Kops/sec)	8.5	12.2	33.3	45.2	54.3	65.4	84.2

Our next experiment measures performance of the integrated (TSDB plus publish/subscribe engine) system focusing on end-to-end latency (time to complete one or more queries over 1M data points) and throughput (publish ops per second). Table 9.2 reports our results focusing on a single query going over 1M data points with increasing scope. Our results show that latency and throughput are increasing with an increasing number of publish-event operations. In practice, such large queries are expected to hurt responsiveness (the time from the occurrence of an event to its publishing). Smaller, more frequent queries should result into longer end-to-end latencies (although the response time of individual event publish operations will improve) and lower aggregate throughput. Experiments with 100 consecutive queries over 10K data points each, publishing a total of 1M events, take 15 sec (compared to 11.88 sec with a single query) and result in a throughput of 67 Kops/sec (compared to 84.2 Kops/sec for a single query). Figures 9.2 and 9.3 depict the corresponding charts of the latency and throughput results.

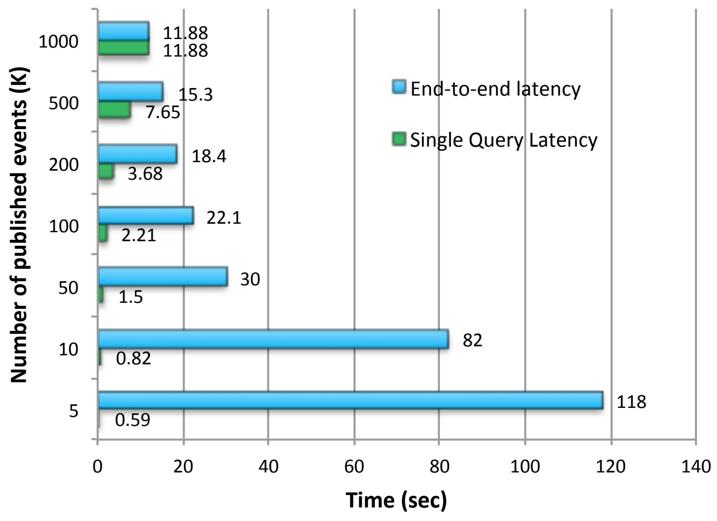


Figure 9.2: Single TSDB query and end-to-end latencies

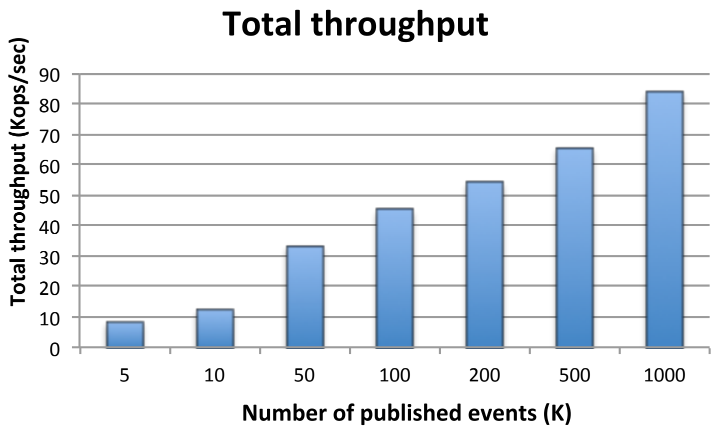


Figure 9.3: Total throughput of a single TSDB query and its publication

As far as the performance of assessing the single raw monitoring events by the Metrics Aggregator component, we rely on the results of our previous work in [Zeginis 2009]. In this work, focusing on monitoring the QoS of Web services using SLAs, we have conducted experiments to evaluate the performance of assessing raw monitoring events against the thresholds, defined as SLOs in a WSLA document. Thus, the results combine the times of parsing the WSLA document to retrieve the corresponding thresholds and assessing a single raw event to determine its criticality (i.e. success, warning, or critical). The experiment results (Figure 9.4) reveal an almost linear relationship between the number of detected monitoring events and the time required to be assessed. These results will be considered for the overall evaluation, analyzed in Section 9.3.

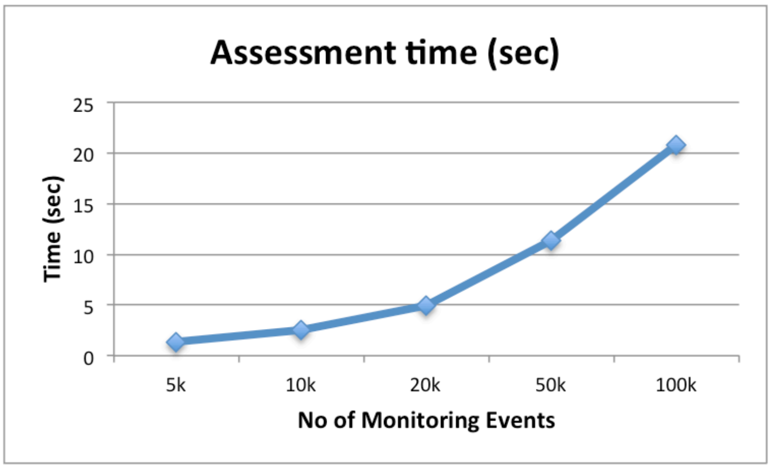


Figure 9.4: Assessment time while increasing the number of monitoring events

9.1.2 Pattern Discovery Algorithm Evaluation

This section describes an experimental evaluation of the proposed pattern discovery algorithm. The aim of this evaluation is to measure: (i) the algorithm's performance and (ii) the algorithm's accuracy, in order to optimize the definition of the considered aggregate metric (i.e., its optimal interval). An event dataset is used, comprising 100k events, including events from Section 2.3's 7 metrics for the traffic management multi-Cloud SBA, collected in a 7-minute time period. The events are provided by: a service-level middleware based on the Astro monitoring tool (service availability, execution time and throughput); the Amazon Cloudwatch PaaS (CPU utilization, data transfer and disk usage metrics for underlying VMs) and a Cloud resource monitor (CPU load, swap and free memory metrics), for monitoring the Flexiant and Openstack VMs. The event timestamp is defined in a millisecond accuracy and during a pre-processing of the event stream we identify five (two 2-size, one 3-size and two 4-size) periodic patterns in it. The 2- and 3-size patterns are periodically injected unchanged in the stream without other mediated events, unlike the 4-size patterns which are interleaved with other random events (one or two events). Thus, taking into consideration this periodicity of the injected patterns, we expect that the pattern discovery algorithm is able to extract them from the monitoring event stream and to identify their λ score. The main goal of this experiment is to discover patterns causing Device Configuration SaaS execution time violations (i.e., violations of E_3 aggregate metric).

The first experiment evaluates the algorithm's raw *relative and absolute accuracy* in discovering all the incorporated and already-known periodic patterns of the 100k-size dataset. The former considers only the five known patterns, while the latter additionally considers their sub-patterns (providing that their λ score is larger than 0), as they can also drive proactive adaptation. For instance, the sub-pattern $\{e_1, e_2\}$ (i.e., sub-pattern of known $\{e_1, e_2, e_4\}$ pattern) is a relevant pattern, as upon its detection we can trigger a suitable adaptation strategy. The bursty or random patterns in the event stream are ignored, as experiments have shown that they map to perfect accuracy results. Thus, the algorithm's *precision* (Equation 9.1), *recall* (Equation 9.2) and *F-measure* (Equation 9.3) are measured while fluctuating the interval size from 4 to 20 events. Precision is the fraction of retrieved patterns that are relevant, while recall is the fraction of relevant patterns that are retrieved. In addition, F-measure can be interpreted as a weighted average of the precision and recall, where an F-measure score reaches its best value at 1 and worst score at 0.

$$\textit{precision} = \frac{relevant\_discovered\_patterns}{total\_discovered\_patterns} \tag{9.1}$$

$$\textit{recall} = \frac{relevant\_discovered\_patterns}{total\_relevant\_patterns} \tag{9.2}$$

$$\textit{F-measure} = \frac{2 * precision * recall}{precision + recall} \tag{9.3}$$

Figure 9.5 shows that *relative precision* is 1 for small intervals and falls while increasing the interval size, while *absolute precision* fluctuates similarly at lower levels (as more irrelevant sub-patterns are discovered). This decrease is normal, as for higher intervals more irrelevant patterns (i.e., larger patterns than already-known ones) are retrieved compared to the total discovered patterns. We must note that the precision starts to fall over 8-size intervals, i.e., above the double of the maximum pattern (4 events).

Figure 9.6 shows that the algorithm's *absolute* recall is 1 for all considered interval sizes, as the predefined event patterns are always discovered, except for 4-size and 6-size intervals where the algorithm fails to discover two and one 4-size patterns respectively, due to interval overlapping. Moreover, *relative* recall is always 1, as the discovered sub-patterns compensate the "lost" relevant patterns (for 4- and 6-size intervals), as they also map to adaptation strategies addressing the whole pattern. Considering these accuracy results, metric E_3's optimal definition is to measure it in intervals containing in average 8 events, as this interval size enables the most accurate pattern discovery results.

The second experiment evaluates the algorithm's execution time, based on: (i) the dataset size; and (ii) the time interval. The former sub-experiment evaluates the algorithm's performance on different dataset sizes for 10-, 20- and 30-size intervals, while the latter is applied on 5k, 20k and 50k-size datasets with an interval fluctuation from 5 to 30 events (upper limit to compute the powerset). Figure 9.8 presents the respective results showing that the algorithm's execution

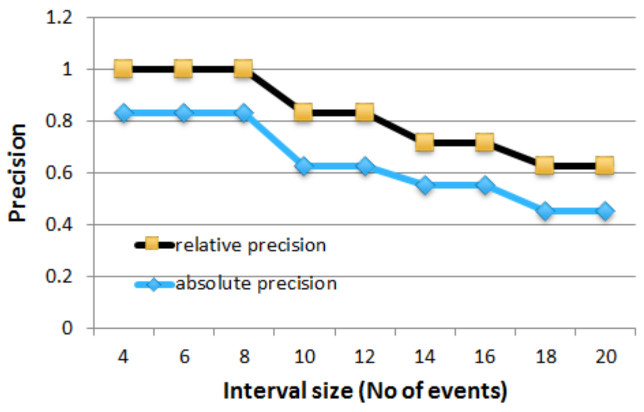


Figure 9.5: Pattern discovery algorithm's precision

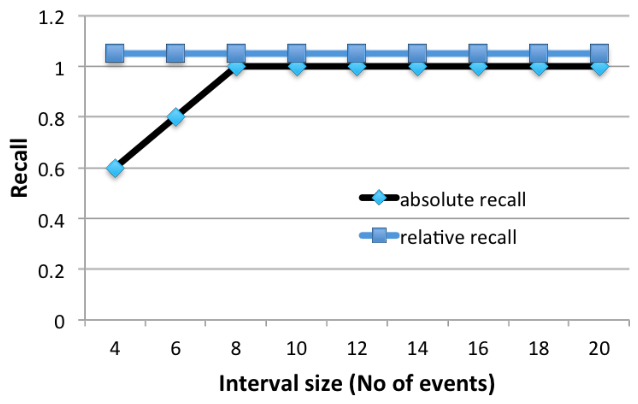


Figure 9.6: Pattern discovery algorithm's recall

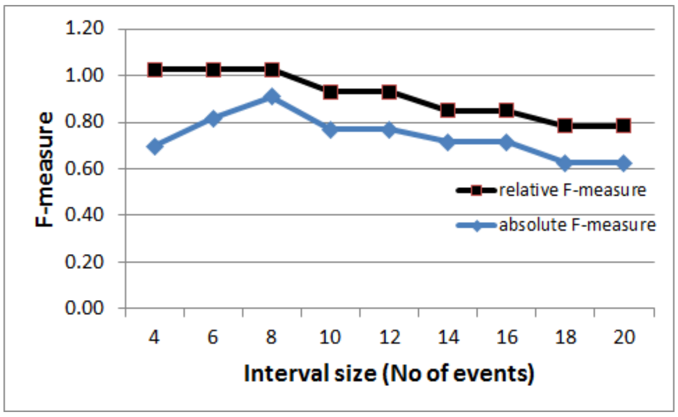


Figure 9.7: Pattern discovery algorithm's F-measure

time almost linearly increases with an increasing dataset size as expected. Larger intervals seem to hurt more the algorithm's performance, due to higher *b*g* products. However, such execution time is acceptable, as this is an offline algorithm not affecting the overall monitoring and adaptation framework performance. Furthermore, the results in Figure 9.9 reveal a changing relation between execution time and interval size; for larger intervals, it increases with a burst over 20-size intervals, due to rapid increase of *b*g* (740 (b=148, g=5) for 30-size interval compared to 92 (b=23, g=4) for the 25-size and 48 (b=12, g=4) for the 20-size intervals), posed by the considered unique sets' high increase. In fact, execution time for the largest interval (30) is more than doubled with respect to 20-size intervals.

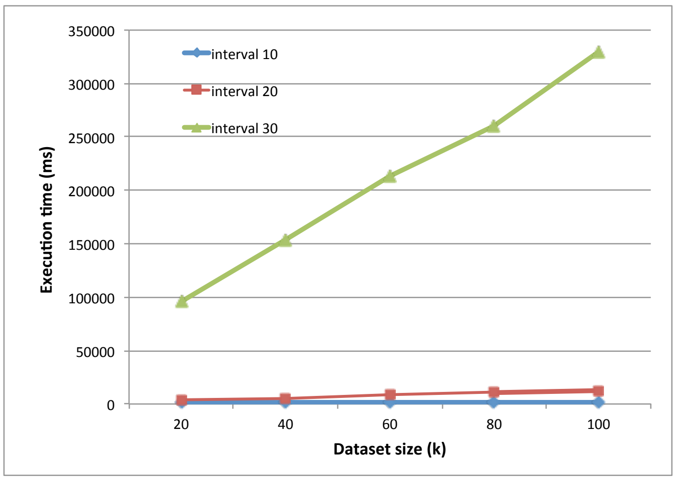


Figure 9.8: Pattern discovery execution time based on the dataset size

9.1.3 Pattern Detection

In this section we evaluate the performance of the pattern detection system. In particular, this experiment measures the pattern detection time of the Esper CEP engine utilized to detect the discovered event patterns at run-time while fluctuating the pattern size. The Esper engine works as a turned upside-down database [Esper 2014]. Instead of storing the data and running queries against stored data, it allows applications to store queries and run the data through. The response from the Esper engine is real-time when conditions that match EPL queries are satisfied. The execution model is thus continuous rather than only when a query is submitted. Thus, even overlapping patterns do not affect the overall performance of the EPD, as the result will be returned as soon as the condition is met. The results in Figure 9.10 verify this conclusion, as for all considered pattern sizes (i.e. the ones returned by the event pattern discovery) the time fluctuates around 8ms.

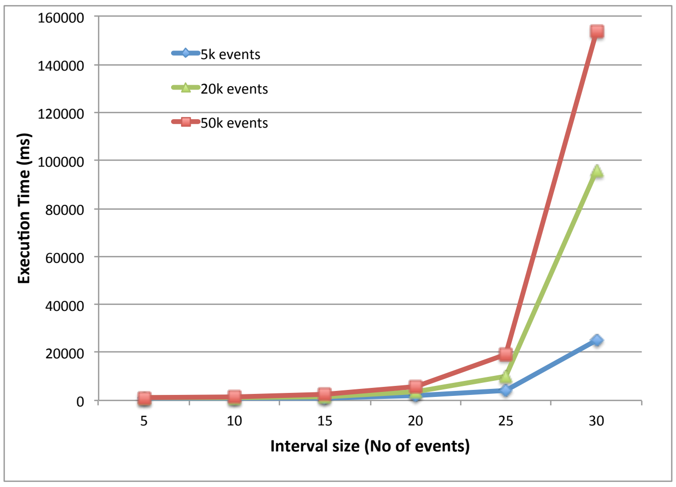


Figure 9.9: Pattern discovery execution time based on the interval size

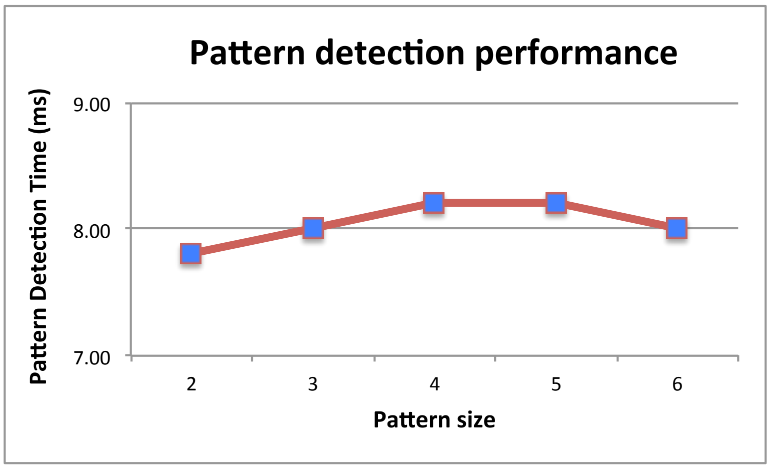


Figure 9.10: Performance of the EPD based on the pattern size

9.2 Adaptation Evaluation

In this section we evaluate the performance and the accuracy of the adaptation engine, which is responsible of enacting suitable adaptation actions upon the detection of a critical event (i.e. reactive adaptation) or a detrimental event pattern verified to cause specific SLO violations (i.e. proactive adaptation) through the ECMAF's incorporated pattern discovery mechanism. Thus, the total duration of the adaptation process comprises the time interval from the occurrence of the critical event or the critical event pattern, to finding and triggering of the corresponding adaptation rule and to enacting the adaptation strategy.

9.2.1 Adaptation Enactment Evaluation

In this section we perform evaluation of the adaptation enactment mechanisms utilized by the ECMAF framework. Especially, this evaluation focuses on the performance of the scaling actions provided by the OpenStack Cloud computing platform. In the Traffic Management running example, this platform is used to deploy the Assessment Web service on. We selected this platform as the base of the adaptation enactment evaluation, as it is a private platform that is easily configurable with many scaling features. However, it is in our plans (as it is analyzed in Section 10.2) to implement or exploit existing adaptation enactment mechanisms, in order to cover as many adaptation actions as possible, including those modeled in the proposed adaptation actions meta-model. These experiments were performed on a machine with Intel®Core i5 480M @ 2.67GHz chipset, 8GB RAM memory, 500GB SATA disk and Windows®operating system. The computation time values are an average of 20 runs.

The following experiments evaluate the scaling actions provided by OpenStack. Whereas traditional applications require more powerful hardware to scale (i.e. "vertical scaling"), cloud-based applications typically request more discrete hardware (i.e. "horizontal scaling"). OpenStack itself is designed to be horizontally scalable[1]. Rather than switching to larger servers, one may procure more servers and simply install identically configured services. Ideally, one scales out and loads balance among groups of functionally identical services (for example, compute nodes or nova-api nodes) that communicate on a message bus.

Determining the scalability of a Cloud and how to improve it is a challenge, as it involves many variables to balance. In fact, no existing solution meets everyone's scalability goals. However, it is helpful to track a number of metrics. Since one can define virtual hardware templates, called "flavors" in OpenStack (see Figure 8.10), she/he can start to make scaling decisions based on the provided flavors. These templates define sizes for memory in RAM, root disk size, amount of ephemeral data disk space available and a number of cores for starters.

Although this starting point meets the initial requirements of the Cloud SBA, during its execution in a vulnerable multi-Cloud environment, new requirements may be posed and SLO violations or critical discovered event patterns may persist over time. Consequently, a common suitable adaptation action for multi-Cloud SBAs is the scaling to higher "flavors", in order to meet the new requirements and constraints posed by the SBA Adaptation Manager.

The following experiments evaluate the performance of horizontal scaling actions, while increasing the number of VM instances (from 1 to 10 instances). The first set of experiments evaluate the performance (i.e. execution time and throughput) of provisioning the supported by the utilized VM "flavors" (i.e. tiny, small and medium). Throughput is measured in a minute scale (i.e. how many VM provisions can be performed within 1 minute).

In particular, these experiments evaluate the performance of the scaling actions, based on the image size and on the VM's "flavor" (see Figure 8.10). For this reason, we are using three different images with increasing sizes ("CIRROS-032, 64" image = 12,6MB, "FEDORA-20, 64" image = 201,1MB and "RHEL-7, 64" image = 415MB). These images are "clean", i.e. they do not install any other software during the provision process. In the future, we plan to perform this evaluation

[1]http://docs.openstack.org/openstack-ops/content/scaling.html

on more powerful machines, supporting all the OpenStack "flavors" and other customized ones, as well as on other larger images. We expect from this experiments to see an almost linear relationship between the performance (execution time) and the number of VM instances.

The individual experiments (Figures 9.11-9.16) for each VM flavor show, in general, a good behavior in terms of performance scalability, even for the largest image (RHEL-7) (almost 2 minutes to provision a VM with an 400MB image on a "tiny" VM). The results reveal that the execution time of provisioning a VM (independently of its flavor) increases almost linearly while increasing the number of VM instances. Similarly, in accordance to the execution time results, throughput (i.e. VM provisions per minute) almost exponentially decreases while increasing the provisioned VM instances. However, the time difference between the execution time of the different images does not reveal any obvious relationship between these two parameters, i.e., image size and number of instances (31,5sec, 62,4sec and 105,35sec to provision one instance with a CIRROS, a FEDORA and a RHEL image respectively). These results are in line with our expectations, expressed before carrying out the experiments.

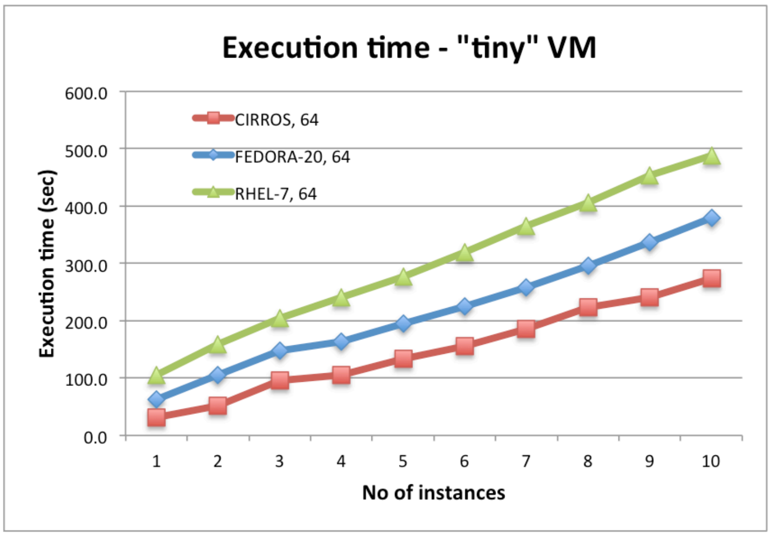


Figure 9.11: VM provision time / No of instances, image size ("Tiny" VM)

Similarly, the second set of experiments evaluates the performance (execu-

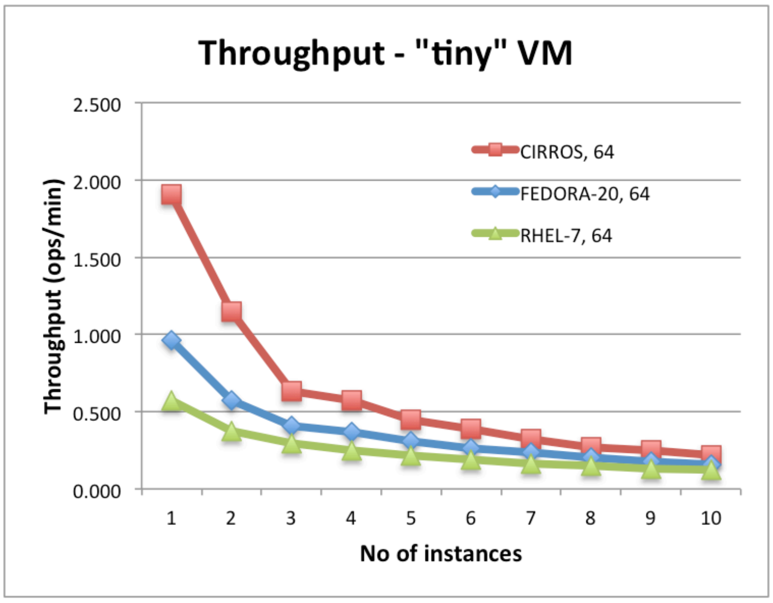


Figure 9.12: VM provision throughput / No of instances, image size ("Tiny" VM)

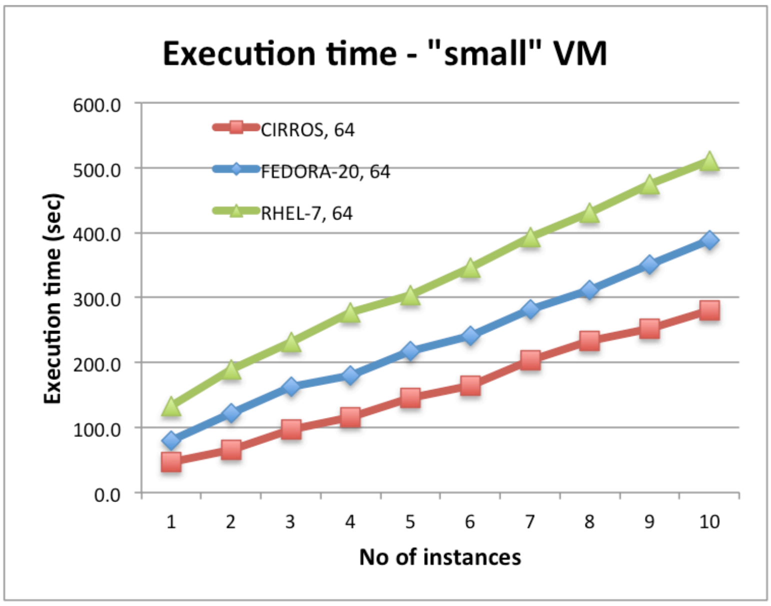


Figure 9.13: VM provision time / No of instances, image size ("Small" VM)

tion time and throughput) of provisioning an incremental number of VM instances (from 1 to 10 instances) for the supported VM "flavors" (i.e. tiny, small and medium). Therefore, these experiments aggregate the results of the previous experiments,

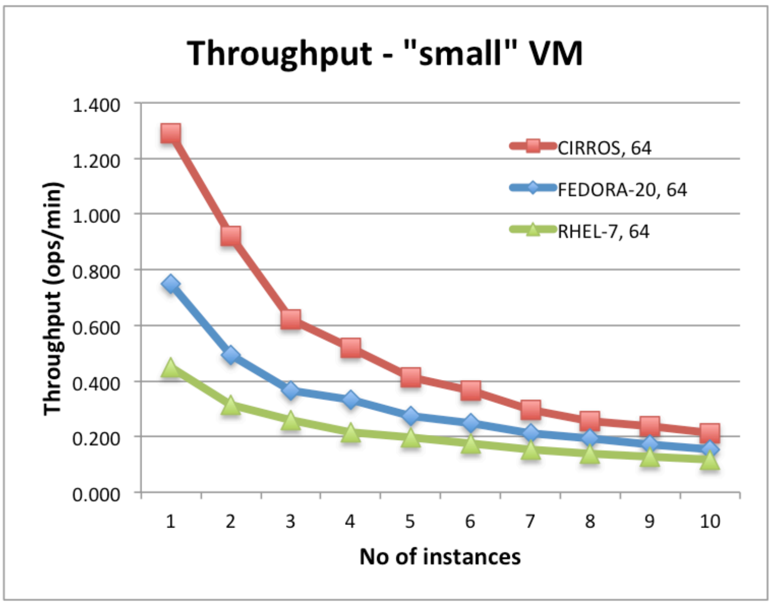


Figure 9.14: VM provision throughput / No of instances, image size ("Small" VM)

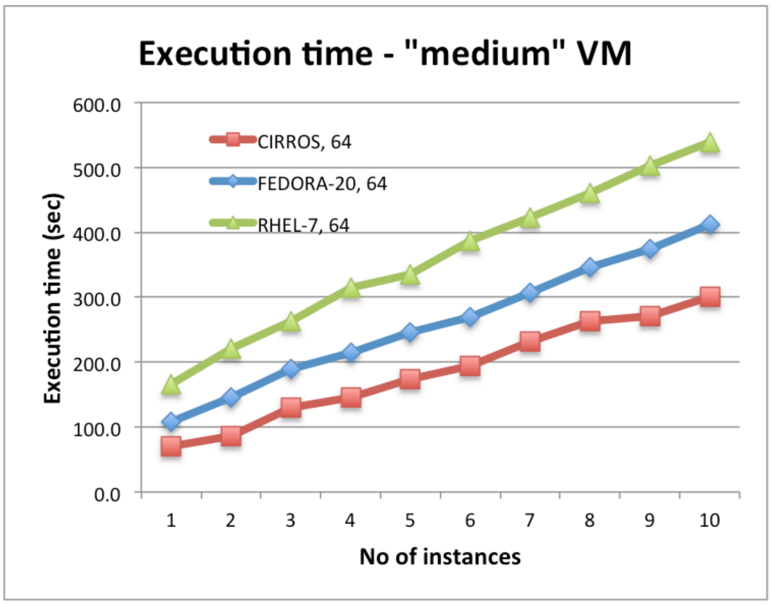


Figure 9.15: VM provision time / No of instances, image size ("Medium" VM)

focusing on the provision performance on the different VM "flavors". The following charts (Figures 9.17-9.22) presents the results of the experiments for the three experimental images (i.e. "CIRROS-032, 64", "FEDORA-20, 64" and "RHEL-7, 64”). For this experiments, we expect that execution times would increase while increasing the instance scope.

The individual experiments for each experimental image show again a good behavior in terms of performance. The results reveal that all the three “clean” images behave similarly, i.e. the execution time increases almost linearly, while increasing the number of provisioned VM instances. However, when we tried

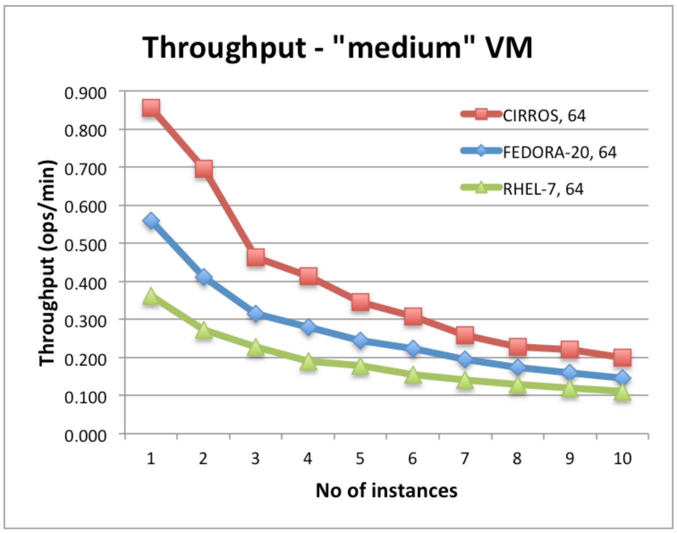


Figure 9.16: VM provision throughput / No of instances, image size (“Medium” VM)

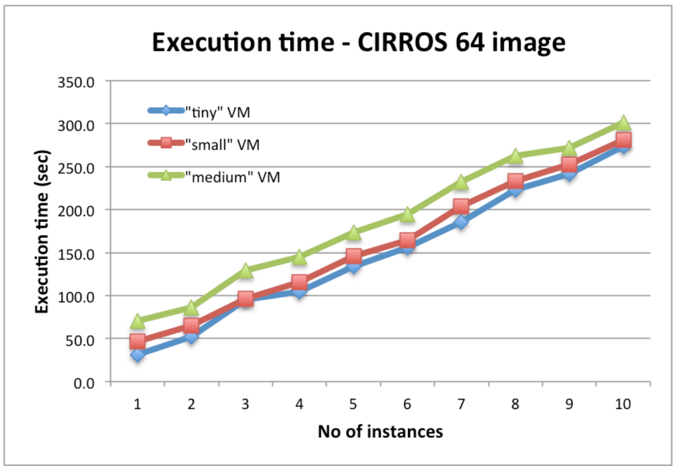


Figure 9.17: VM provision time / No of instances, VM flavor (CIRROS-032 image)

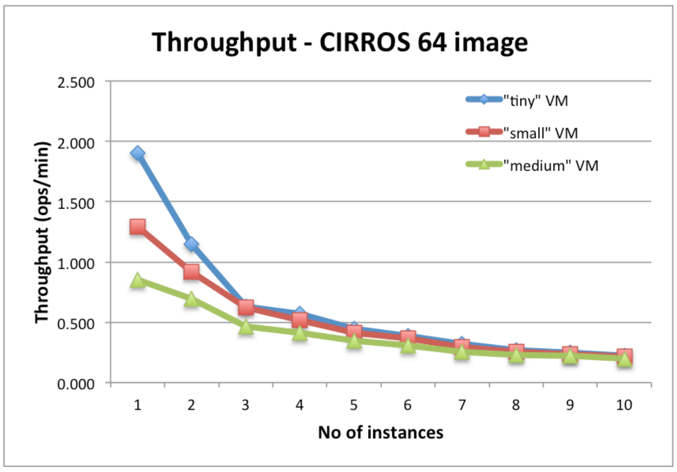


Figure 9.18: VM provision throughput / No of instances, VM flavor (CIRROS-032 image)

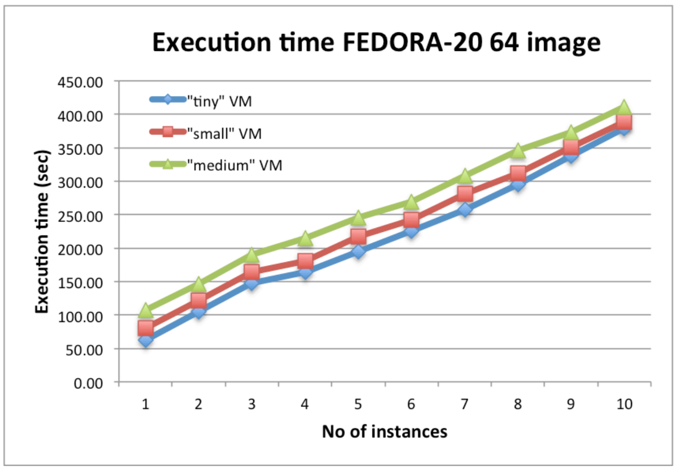


Figure 9.19: VM provision time / No of instances, VM flavor (FEDORA-20 image)

to provision VMs with default images provided by various Linux providers (e.g. openSUSE[2], Ubuntu[3] and others), that are downloading a couple of packages during the installation, we got very large times (over 8 minutes for a single instance). Throughput, again, exponentially decreases while increasing the number of VM instances. As in the previous set of experiments, the results are in line with our expectations.

[2]http://www.opensuse.org
[3]http://www.ubuntu.com

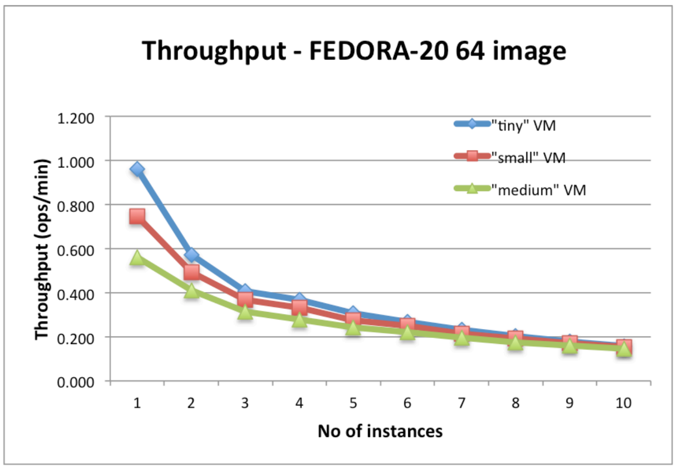


Figure 9.20: VM provision throughput / No of instances, VM flavor (FEDORA-20 image)

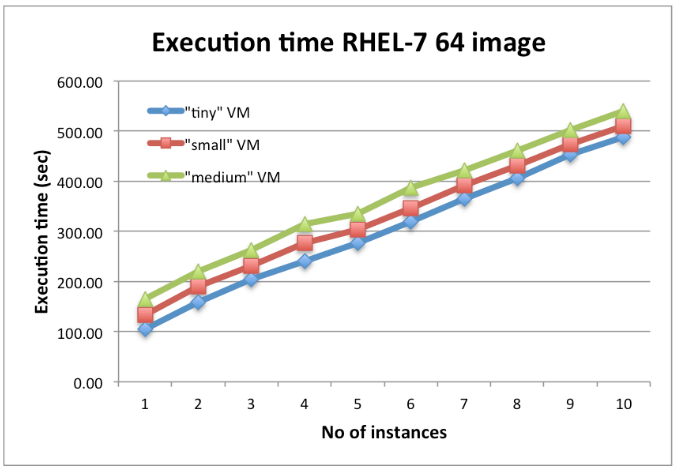


Figure 9.21: VM provision time / No of instances, VM flavor (RHEL-7 image)

9.3 Overall Evaluation

This section elaborates on the overall evaluation of the ECMAF framework. In order to perform this evaluation we take into consideration the results of the experiments that were conducted in the previous sections. The overall evaluation discerns two cases. The first one (Figure 9.23) deals with reactive adaptation, combining the results of getting the monitoring events, assessing them and storing them to the KairosDB and enacting a suitable scaling action. The second one (Fig-

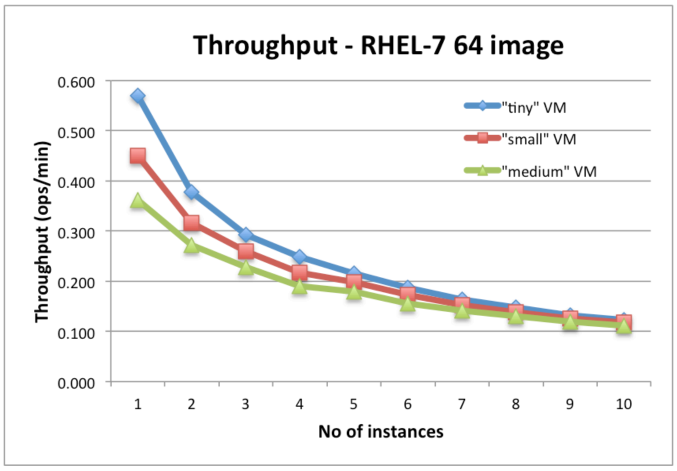


Figure 9.22: VM provision throughput / No of instances, VM flavor (RHEL-7 image)

ure 9.24) deals with proactive adaptation and thus combines the following times: (i) the time of getting the monitoring results and storing them to the KairosDB, (ii) the time of querying the KairosDB to get an event stream, (iii) the time of discovering the detrimental event patterns within the considered event stream, (iv) the time of detecting such a pattern; and (v) the time of enacting a suitable scaling action.

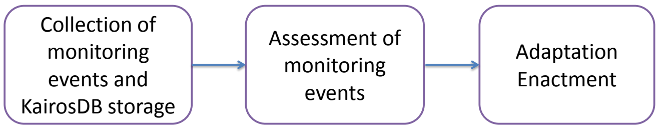

Collection of monitoring events and KairosDB storage
Assessment of monitoring events
Adaptation Enactment

Figure 9.23: The reactive adaptation path of ECMAF

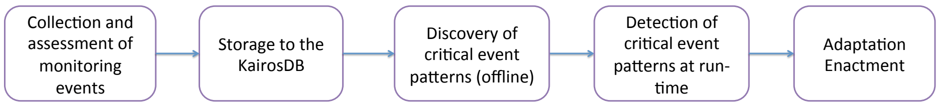

Collection and assessment of monitoring events
Storage to the KairosDB
Discovery of critical event patterns (offline)
Detection of critical event patterns at run-time
Adaptation Enactment

Figure 9.24: The proactive adaptation path of ECMAF

The next experiments focus on the overall (end-to-end) performance of the ECMAF, both for the reactive (Figures 9.25 and 9.26) and proactive (Figures 9.27

and 9.28) adaptation scenarios, meticulously analyzed in Section 2.4. These two scenarios, based on the traffic management running example, describe the overall functionality of the proposed framework and clearly indicates its efficient handling in monitoring and adapting SBAs deployed on multiple Clouds.

The first experiment deals with the ECMAF's performance on the reactive adaptation scenario described in Section 2.4. This scenario involves the immediate handling of a response time violation of the Assessment service, hosted on a OpenStack[4] VM. Thus, three times should be taken into account for the overall performance, as illustrated in Figure 9.23: (i) the time to collect and assess the monitoring events, (ii) the time to publish them to the Adaptation Engine and store them into the TSDB; and (iii) the adaptation enactment time, i.e. the time of the "scaling-out" action to deploy two more "tiny" VM instances with RHEL OS, in order to address the demanding load.

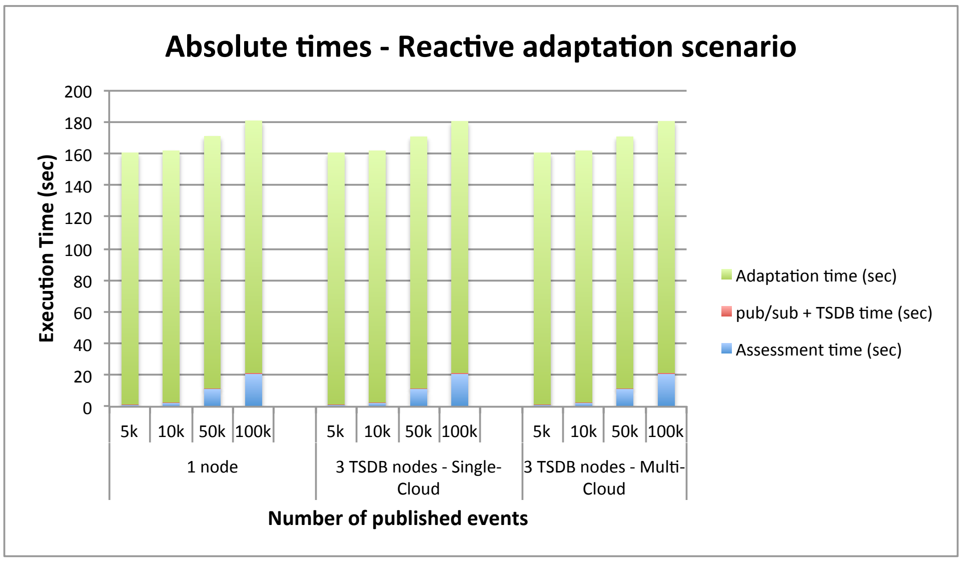


Figure 9.25: Time distribution in a reactive adaptation scenario

The results in Figures 9.25 and 9.26 show that the majority of the end-to-end time is consumed in executing the adaptation action. Independently of the TSDB setup, the publishing time and the time to store the monitoring events into the

[4]http://www.openstack.org

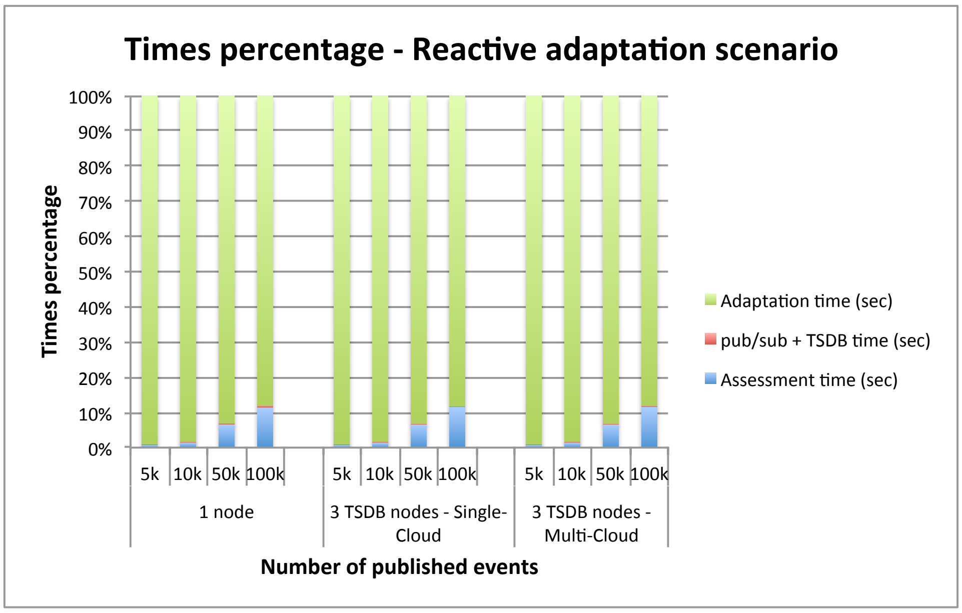


Figure 9.26: Percentage time distribution in a reactive adaptation scenario

TSDB, as well as the assessment time, do not significantly affect the overall performance time in handling this reactive adaptation scenario. Only when publishing a great amount of events (50k and 100k) the assessment time starts to have a substantial impact on the overall performance.

The second experiment deals with the ECMAF's performance on the proactive adaptation scenario described in Section 2.4. This scenario involves the handling of the critical event pattern consisting of a CPU usage warning violation of the VM hosting the Monitoring service and available free memory violation of the same VM. This event pattern has been discovered and related to cause a violation of the execution time SLO of the Monitoring service. In this scenario five times should be taken into account for the overall performance, as illustrated in Figure 9.24: (i) the time to collect and assess the monitoring events, (ii) the time to store them into the TSDB, (iii) the pattern discovery time, (iv) the pattern detection time; and (v) the adaptation enactment time, i.e. the time of provisioning one more instance of a "medium" Openstack VM with Fedora OS, thus preventing the expected execution time violation.

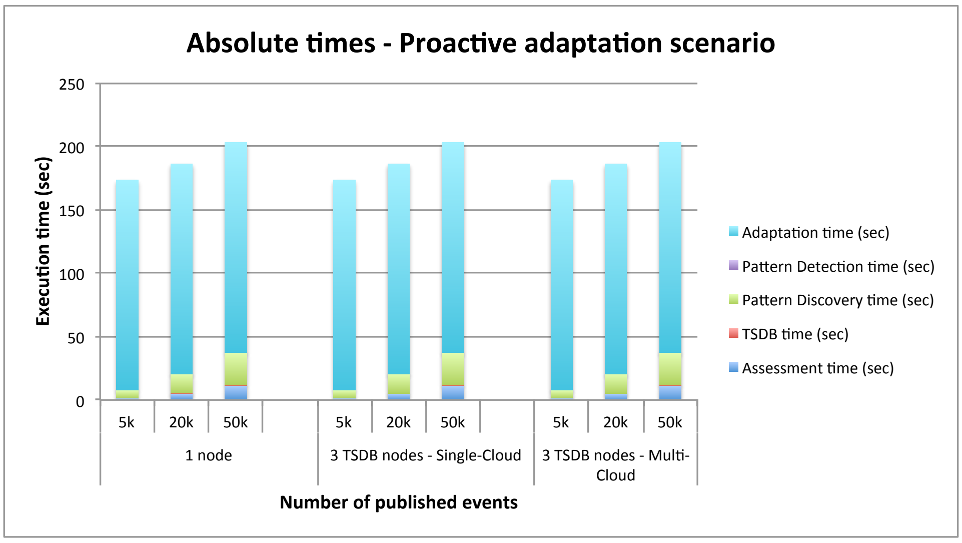


Figure 9.27: Time distribution in a proactive adaptation scenario

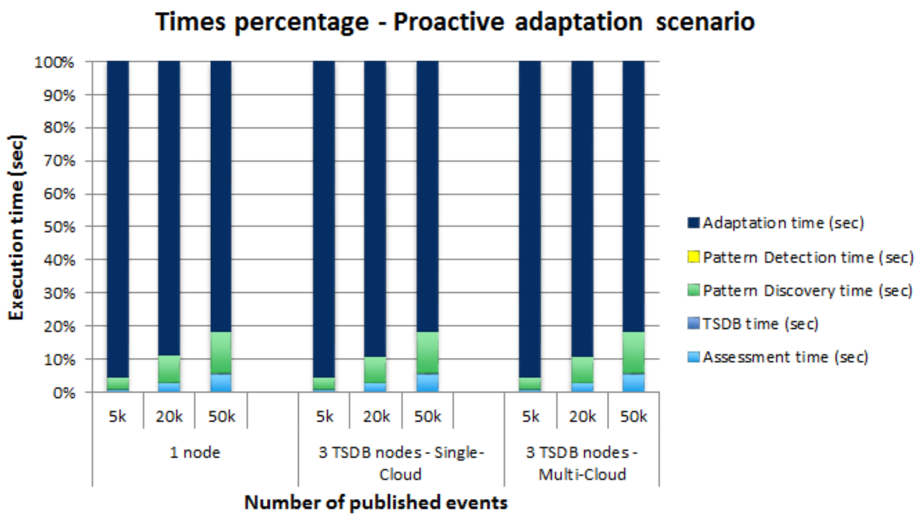


Figure 9.28: Percentage time distribution in a proactive adaptation scenario

The results in Figures 9.27 and 9.28 show that the majority of the end-to-end time is consumed again in executing the adaptation action (“scaling-out” action). Similarly to the reactive adaptation scenario, the times to store the monitoring events into the TSDB and to detect the already discovered pattern, do not significantly affect the overall performance time in handling this proactive adaptation scenario. Only when publishing a great amount of events (20k and 50k) the assessment time and the pattern discovery time start to have a substantial impact on the overall performance.

9.4 Conclusions

To sum up, this chapter evaluates the performance, as well as the accuracy of the individual components of the ECMAF framework. The first two parts of the chapter focus on the evaluation of the individual components of the ECMAF's monitoring and adaptation engines respectively, while the final part focuses on the overall evaluation of the proposed framework. The overall results are in general satisfactory regarding both regarding the reactive adaptation and proactive adaptation paths. However, they are amenable to improvements and optimizations, as they are described in the next chapter.

Chapter 10

Conclusions and Future Research

Contents

10.1 Synopsis of Contributions

The primary objective of the research behind this dissertation is to motivate the need for monitoring and adapting Service-based Applications (SBAs) deployed on multiple Clouds, across all the functional layers of the SOA and Cloud stack. This need led to the proposal of a new innovative monitoring and adaptation framework for Cloud-based SBAs, named ECMAF. This framework advances service research by addressing these two processes in a cross-layer way and additionally supports many challenging features in service monitoring and adaptation, such as event pattern discovery and proactive adaptation, in a unified and integrated manner.

In particular, in this dissertation we identify the major problems stemming from the lack of cross-layer monitoring and adaptation techniques addressing all the functional layers of the SOA and Cloud context. The conducted literature review revealed a gap in this service area, due to the current mainly fragmented approaches. Thus, it is crucial to explicitly relate different conceptual elements of the SBAs and the exploited Cloud artifacts to interrelate them and discover cross-cutting event patterns leading to specific SLO violations. To achieve this, there is a need to develop high-level meta-models, which relate: (i) elements of the SBA with the underlying infrastructure (local or virtualized), (ii) monitoring events with suitable adaptation strategies and the corresponding mechanisms, that are available at different layers; and (iii) monitoring events with their source components. Based on these models, novel cross-cutting approaches can propagate aligned monitoring events and reflect the relations and the impact of the adaptation activities on the different layers.

The proposed ECMAF framework is able to detect monitoring events across the SBA and Cloud functional layers and to derive suitable adaptation strategies using the incorporated Adaptation Manager. ECMAF integrates monitoring mechanisms within each Cloud layer and across Cloud providers. A Monitor Manager collects the timely synchronized monitoring events and stores them in a time-series database (TSDB), which is responsible for the raw measurements' storage and the extraction of aggregated values, which, in turn, are utilized by the Metric Aggregator and the Pattern Discoverer components. The former one assesses the aggregate metrics against the thresholds defined in the SLOs of the corresponding SLA document (expressed in the WSLA language), while the latter one discovers frequent patterns of monitoring events leading to specific SLO violations. This pattern discovery technique confers proactive capabilities to the ECMAF, while the efficient rules management reinforces its reactive adaptation capabilities. The Monitor Manager also propagates important events to the interested subscribers such as the incorporated Adaptation Rule Manager. This component is responsible for processing the output of the assessment process and the detected critical event patterns. Then, it either triggers a simple adaptation rule for addressing the single SLO violation or triggers a more complex adaptation, addressing detected critical event pattern, thus proactively adapting the system before the actual SLO violations occurs.

The proposed architecture uses an event meta-model, a component meta-model and an adaptation actions meta-model for the description of the monitoring events, the SBA's components and the supported adaptation actions respectively. The event meta-model describes the most common monitoring event types and patterns that occur during the Cloud SBA execution. It is generic enough and extensible to incorporate any other event type defined by domain-specific service providers. The component meta-model describes the source components for each event type that constitute the SBA system, as well as the dependencies among the qualitative and quantitative attributes of a SBA and the monitored source components producing events at run-time. The adaptation actions meta-model describes the adaptation actions that can be enacted on each one of the functional layers of a SBA. All these models facilitate: (i) the correlation between events which lead to critical performance violations or faults, (ii) the mapping of critical events or critical event patterns to suitable adaptation actions, (iii) the description of cross-layer dependencies between the various system components; and (iv) the assessment of the current system state. Collectively, they capture all information required for optimizing and adapting applications deployed over multiple Clouds.

In a nutshell, the main benefits of the proposed monitoring and adaptation framework for Cloud-based SBAs are the following:

- **Distributed workload:** The monitoring workload is distributed across the functional layers. Furthermore, monitoring and adaptation may be delegated to different components based on the monitored metrics and the enacted adaptation actions.

- **Extensibility:** ECMAF can integrate new monitoring and adaptation techniques with the existing ones, while preserving its functionality and integrity.

- **Cross-layer and Multi-Cloud capability:** The framework is able to support all Cloud and SOA layers.

- **Pro-active and reactive adaptation:** Discovery of warning event patterns enables pro-active adaptation, while the efficient rules management reinforces both pro-active and reactive adaptation capabilities.
- **Functional and non-functional (QoS)** properties are supported.

As far as the applicability of the ECMAF framework is concerned, it can have a substantial impact to SOA stakeholders, especially to those who resort in multi-Cloud deployments of their SBAs. This framework enables them to have a clear view of how their applications perform in such distributed virtualized environments, where they do not have direct access to the underlying resources (i.e. hosting machines). Moreover, the incorporated pattern discovery mechanism assists them in performing a root cause analysis in the monitoring events stream and identifies the main source of malfunctions in this multi-tier environment. The time synchronization techniques and the time-series database (TSDB) utilized by the Monitoring Engine enable application developers to track the monitoring execution history, maintaining the actual order of the monitoring events and thus reliably discovering their causal relationships. Concerning SBA adaptation managers, they can take advantage of the proposed framework, as it provides semi-automatic adaptation capabilities, that can be exploited to define the desired mapping of simple SLO violations to suitable adaptation actions provided by the available adaptation mechanisms. Additionally, the incorporated event Pattern Discoverer enables injecting proactiveness in a SBA adaptation engine via mapping event patterns to respective adaptation strategies (involving other lower level adaptation actions) through an automatic mapping mechanism. This mapping lowers the adaptation cost and at the same time makes SBAs more attractive to the potential customers.

The applicability and usability of the proposed framework in realistic settings is backed up by the experimental evaluation that we conducted. The experiments prove that all phases of the monitoring and adaptation process scale well and can efficiently manage both simple and complex scenarios. The performance evaluation of the cross-layer monitoring framework in different deployment settings shows that performance of TSDB scales while increasing the number of storage servers and has minimal impact in a multi-Cloud scenario based on the traffic management running example, involving different Cloud providers. As far as the performance and accuracy of the pattern discovery process is concerned, the results show that they are optimal when suitably configured by appropriately selecting the dataset and interval (window) size. Moreover, the evaluation of the adaptation process reveal satisfactory results both in terms of reactive and of proactive adaptation features. Finally, the overall evaluation, combining the results of the monitoring and adaptation experiments, shows that ECMAF can efficiently handle the individual critical events, as well as the critical event patterns causing specific SLO violations.

10.2 Directions for Future Work

As for future work, we are planning to optimize the proposed pattern discovery technique in terms of accuracy and performance, so as to enable efficient handling of all the monitored metrics within the optimized time ranges, as well as to support larger intervals to accommodate the huge amount of monitoring events detected in large-scale systems. The Adaptation Engine will be enhanced, so as to be able to address more complex adaptation strategies spanning across different functional layers. Moreover, we intend to employ a more comprehensive rule processing system, that will extract semantically equivalent rules from the simple adaptation rules defined by the SBA Adaptation Manager.

The three proposed meta-models are extensible enough to incorporate new UML classes that will express additional aspects of the constantly evolving Cloud environment or will be dictated by the SBA evolution process. Consequently, the event meta-model could be enhanced to address new metrics supported by the exploited monitoring mechanisms. Respectively, the component meta-model is amenable to many additions based on the virtual resources provided by the constantly evolving offerings of the Cloud providers. Similarly, the adaptation actions meta-model will be enriched with new adaptation actions provided by the available Cloud adaptation enactment mechanisms.

As far as the monitored properties supported by the ECMAF framework are concerned, we plan to exploit WSSL language [Baryannis and Plexousakis 2014], which is a novel service specification language based on the fluent calculus, addressing issues related to the frame, ramification and qualification problems. One of its main applications is in specifying pre-conditions and post-conditions of Web services and verifying whether they are satisfied or not. Thus, we can support more comprehensive adaptation rules addressing not only QoS aspects, but also functional characteristics of the monitored SBAs.

Furthermore, as part of the work presented in [Zeginis et al. 2012a] and especially the extensions proposed for OWL-Q language [Kritikos and Plexousakis 2006], we plan to employ OWL-Q with new aspects addressing all the functional and non-functional properties of an SBA and the penalties posed upon their violation. Moreover, another future challenge is the definition of composite OWL-Q metric formulas, that can measure more complex aggregate metrics of the monitored SBAs. These formulas can infer metric values for composite Web services, exploiting the corresponding values of the constituted simple Web services [Zeginis 2009].

Regarding the existing Monitor Manager, we plan to enrich the cross-layer monitoring mechanism by developing a reporting tool that will enable system administrators and business experts to observe the problematic situations and perform statistical analysis. This analysis will also facilitate the Pattern Discoverer by injecting to the existing logic-based mechanism statistical results, that can be used to limit the considered sets in extracting the critical event patterns. In the long-term, we will exploit semantic descriptions to automate the various activities of our approach and to derive new knowledge in terms of monitoring events, concerning external services.

Finally, we are also planning larger-scale end-to-end multi-Cloud experiments involving long-running SBAs, other than the traffic management running example. Based on these results, we expect to optimize the performance and accuracy of the ECMAF framework. These experiments would also involve functional aspects of the measured SBAs, supported by the aforementioned WSSL language, as well as event patterns combining functional and QoS properties.

Acronyms

ADT	Average Daily Traffic
AOP	Aspect-Oriented Programming
API	Application Programming Interface
AQI	Air Quality Index
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
CEP	Complex Event Processing
ECA	Event-Condition-Action
EPD	Event Pattern Detector
EPL	Esper Event Processing Language
DaaS	Data-as-a-Service
DRL	Drools Rule Language
DSL	Domain Specific Language
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
LBPD	Logic-based Pattern Discovery
minsup	Minimum Support

MA	Monitoring and Adaptation
MaaS	Monitoring-as-a-Service
MDT	Metric Derivation Tree
NaaS	Network-as-a-Service
OS	Operating System
PaaS	Platform-as-a-Service
PS	Prioriry Score
RTML	Real Time Markup Language
QoS	Quality of Service
SaaS	Software-as-a-Service
SBA	Service-based Application
SCC	Service Composition and Coordination
SECaaS	Security-as-a-Service
SN	Service Network
SI	Service Infrastructure
SLA	Service-level Agreement
SLO	Service-level Objective
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
THI	Temperature-Humidity Index
TM	Traffic Management

TSDB	Time-series Database
UML	Unified Modeling Language
VM	Virtual Machine
VSP	Virtual Service Platform
WAR	Web Application ARchive
WSLA	Web Service Level Agreement
XaaS	Everything-as-a-Service
XML	XML Meta-data Interchange

Bibliography

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487–499.

Alcaraz Calero, J., König, B., and Kirschnick, J. 2012. Using cross-layer techniques for communication systems. In *Premier reference source*. Igi Global, Chapter Cross-layer monitoring in Cloud computing.

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M. 2007. Web services agreement specification (ws-agreement). Tech. rep. March.

Artikis, A., Sergot, M., and Paliouras, G. 2010. A logic programming approach to activity recognition. In *Proceedings of the 2Nd ACM International Workshop on Events in Multimedia*. EiMM '10. ACM, New York, NY, USA, 3–8.

Artikis, A., Sergot, M. J., and Paliouras, G. 2012. Run-time composite event recognition. In *DEBS*, F. Bry, A. Paschke, P. T. Eugster, C. Fetzer, and A. Behrend, Eds. ACM, 69–80.

Artikis, A., Weidlich, M., Gal, A., Kalogeraki, V., and Gunopulos, D. 2013. Self-adaptive event recognition for intelligent transport management. In *BigData Conference*. IEEE, 319–325.

Barbon, F., Traverso, P., Pistore, M., and Trainotti, M. 2006. Run-time Monitoring of Instances and Classes of Web Service Compositions. In *ICWS*. IEEE, 63–71.

Baresi, L. and Guinea, S. 2005. Dynamo: Dynamic Monitoring of WS-BPEL Processes. In *ICSOC*. Springer, 478–483.

Baresi, L., Guinea, S., and Pasquale, L. 2007. Self-healing BPEL Processes with Dynamo and the JBoss Rule Engine. In *ESSPE '07 in conjunction with the 6th ESEC/FSE joint meeting*. ACM, 11–20.

Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., Papaioannou, A., Plexousakis, D., and Zeginis, C. 2013a. Lifecycle Management of Service-based Applications on Multi-Clouds: A Research Roadmap. In *MultiCloud*.

Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., Papaioannou, A., Plexousakis, D., and Zeginis, C. 2013b. Lifecycle management of service-based applications on multi-clouds: a research roadmap. In *Proceedings of the international workshop on Multi-cloud applications and federated clouds*. ACM, 13–20.

Baryannis, G. and Plexousakis, D. 2014. Fluent calculus-based semantic web service composition and verification using wssl. In *Service-Oriented Computing ICSOC 2013 Workshops*. Lecture Notes in Computer Science Series, vol. 8377. Springer International Publishing, 256–270.

Benbernou, S., Cavallaro, L., Hacid, M. S., Kazhamiakin, R., Kecskemeti, G., Pazat, J.-L., Silvestri, F., Uhlig, M., and Wetzstein, B. 2008. PO-JRA-1.2.1, State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs. Tech. rep., S-cube. July.

Bettini, C., Wang, X. S., Jajodia, S., and Lin, J.-L. 1998. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Trans. Knowl. Data Eng. 10,* 2, 222–237.

Blair, G., Bencomo, N., and France, R. B. 2009. Models@ run.time. *Computer 42,* 10, 22–27.

Bo, H. 2011. Stream Database Survey. Tech. rep., University of Waterloo, Computer Science Department.

Bratanis, K., Dranidis, D., and Simons, A. J. H. 2011. Slas for cross-layer adaptation and monitoring of service-based applications: a case study. In *QASBA*. ACM International Conference Proceeding Series. ACM, 28–32.

Carzaniga, A., Papalini, M., and Wolf, A. L. 2011. Content-based publish/subscribe networking and information-centric networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*. ICN '11. ACM, 56–61.

Chaturvedi, S., Prasad, K. H., Faruquie, T. A., Chawda, B., Subramaniam, L. V., and Krishnapuram, R. 2013. Automating pattern discovery for rule based data standardization systems. In *ICDE*. IEEE Computer Society, 1231–1241.

Cristian, F. 1989. Probabilistic clock synchronization. *Distributed Computing 3*, 146–158.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. 2005. Colombo: Lightweight middleware for service-oriented computing. *IBM Systems Journal 44,* 4, 799–820.

Danylevych, O., Leymann, F., and Nikolaou, C. 2011. A framework of views on service networks models. In *EOMAS*. Lecture Notes in Business Information Processing Series, vol. 88. Springer, 21–34.

Deri, L., Mainardi, S., and Fusco, F. 2012. tsdb: A Compressed Database for Time Series. LNCS. Springer, 143–156.

Dib, D., Parlavantzas, N., and Morin, C. 2012. Towards multi-level adaptation for distributed operating systems and applications. In *ICA3PP (2)*. Lecture Notes in Computer Science Series, vol. 7440. Springer, 100–109.

Dranidis, D., Kourtesis, D., and Ramollari, E. 2007. Formal verification of web service behavioural conformance through testing.

Dranidis, D., Metzger, A., and Kourtesis, D. 2010. Enabling proactive adaptation through just-in-time testing of conversational services. In *ServiceWave*. Lecture Notes in Computer Science Series, vol. 6481. Springer, 63–75.

Drools. 2014. Drools expert user guide. Tech. rep.

Durand, J., Otto, A., Pilz, G., and Rutt, T. 2014. Cloud application management for platforms. *OASIS*.

Dustdar, S. 2014. Principles and methods for elastic computing. In *CBSE'14, Proceedings of the 17th International ACM SIGSOFT Symposium on Component-Based Software Engineering (part of CompArch 2014)*. 1–2.

Esper. 2014. Esper reference version 5.0.0.

Farrell, A., Sergot, M., Salle, M., and Bartolini, C. 2004. Using the Event Calculus for the Performance Monitoring of Service-Level Agreements for Utility Computing. In *WEC*. Vol. 6. Citeseer.

Fernandez, E. B. 2012. Introduction to the special issue on recent advances in web services. *Future Internet 4,* 3, 618–620.

Freitas, A. L., Parlavantzas, N., and Pazat, J.-L. 2011. Cost reduction through sla-driven self-management. In *ECOWS*. IEEE, 117–124.

Fugini, M. and Siadat, H. 2009. Sla contract for cross-layer monitoring and adaptation. In *Business Process Management Workshops*. Lecture Notes in Business Information Processing Series, vol. 43. Springer, 412–423.

Garefalakis, P. and Magoutis, K. 2012. Improving datacenter operations management using wireless sensor networks. In *iThings*.

Gjørven, E., Rouvoy, R., and Eliassen, F. 2008. Cross-layer self-adaptation of service-oriented architectures. In *MW4SOC*. ACM, 37–42.

Guinea, S., Kecskemeti, G., Marconi, A., and Wetzstein, B. 2011. Multi-layered monitoring and adaptation. Lecture Notes in Computer Science Series, vol. 7084. Springer, 359–373.

Gusella, R. and Zatti, S. 1986. An election algorithm for a distributed clock synchronization program. 364–371.

Hamilton-Wright, A. and Stashuk, D. W. 2008. Statistically based pattern discovery techniques for biological data analysis. In *Applications of Computational Intelligence in Biology*. Studies in Computational Intelligence Series, vol. 122. Springer, 3–31.

Hasselmeyer, P., Katsaros, G., Koller, B., and P., W. 2012. Using cross-layer techniques for communication systems. Premier reference source. Igi Global, Chapter Cloud monitoring.

Hellerstein, J. L., Ma, S., and Perng, C.-S. 2002. Discovering actionable patterns in event data. *IBM Systems Journal 41,* 3, 475–493.

Hielscher, J., Kazhamiakin, R., Metzger, A., and Pistore, M. 2008. A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In *ServiceWave* (2008-12-14), P. Mahonen, K. Pohl, and T. Priol, Eds. Lecture Notes in Computer Science Series, vol. 5377. Springer, 122–133.

Horn, P. 2001. Autonomic Computing: IBM's Perspective on the State of Information Technology. Tech. rep.

Huhns, M. and Singh, M. 2005. Service-oriented computing: key concepts and principles. *IEEE Internet Computing 9,* 1, 75–81.

Inzinger, C., Hummer, W., Satzger, B., Leitner, P., and Dustdar, S. 2014. Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems. *Software: Practice and Experience 44,* 7, 805–822.

Inzinger, C., Satzger, B., Leitner, P., Hummer, W., and Dustdar, S. 2013. Model-based adaptation of cloud computing applications. In *MODELSWARD*. 351–355.

Jiang, S., Hallsteinsen, S., and Lie, A. 2011. An experimental facility for cross-layer adaptation of service oriented distributed systems. In *NIK*. Tapir Akademisk Forlag, 97–108.

Josh, A. 2012. Types of cloud computing: Private, public and hybrid clouds.

Karastoyanova, D. and Leymann, F. 2009. BPEL'n'Aspects: Adapting Service Orchestration Logic. In *ICWS* (2009-11-12). IEEE.

Karp, R. M. and Rabin, M. . 1987. Efficient randomized pattern-matching algorithms. *IBM Journal Research and Development 31,* 2, 249–260.

Kazhamiakin, R., Pistore, M., and Zengin, A. 2009a. Cross-layer adaptation and monitoring of service-based applications. In *MONA+ Workshop of ICSOC/Service-Wave*. LNCS Series, vol. 6275. 325–334.

Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., and Leymann, F. 2009b. 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. Springer-Verlag, 395–404.

Keller, A. and Ludwig, H. 2003. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management 11,* 1, 57–81.

Keogh, E., Chu, S., Hart, D., and Pazzani, M. 1993. Segmenting Time Series: A Survey and Novel Approach. In *Data mining in Time Series Databases*. Publishing Company, 1–22.

Kongdenfha, W., Motahari-Nezhad, H. R., Benatallah, B., Casati, F., and Saint-Paul, R. 2009. Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters. *IEEE Trans. Serv. Comput. 2*, 94–107.

Konsolaki, K. 2012. Monitoring qos for composite web services. M.S. thesis, University of Crete, Greece.

Kopetz, H. and Ochsenreiter, W. 1987. Clock synchronization in distributed real-time systems. *IEEE Trans. Computers 36,* 8, 933–940.

Kritikos, K. and Plexousakis, D. 2006. Semantic QoS Metric Matching. In *ECOWS*. IEEE Computer Society, Zurich, Switzerland.

Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21,* 7, 558–565.

Laxman, S., Sastry, P. S., and Unnikrishnan, K. P. 2007. Discovering frequent generalized episodes when events persist for different durations. *IEEE Trans. Knowl. Data Eng. 19,* 9, 1188–1201.

Leitner, P., Hummer, W., and Dustdar, S. 2013. Cost-based optimization of service compositions. *IEEE T. Services Computing 6,* 2, 239–251.

Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S. 2010a. Monitoring, prediction and prevention of sla violations in composite services. In *ICWS*. IEEE Computer Society, 369–376.

Leitner, P., Wetzstein, B., Karastoyanova, D., Hummer, W., Dustdar, S., and Leymann, F. 2010b. Preventing sla violations in service compositions using aspect-based fragment substitution. In *ICSOC*. Lecture Notes in Computer Science Series, vol. 6470. 365–380.

Li, H.-F., Lee, S.-Y., and Shan, M.-K. 2004. An efficient algorithm for mining frequent itemsets over the entire history of data streams.

Magoutis, K., Devarakonda, M. V., Joukov, N., and Vogl, N. G. 2008. Galapagos: Model-driven discovery of end-to-end application - storage relationships in distributed systems. *IBM Journal of Research and Development 52,* 4-5, 367–378.

Mahbub, K. and Spanoudakis, G. 2007. *Monitoring WS-Agreements: An Event Calculus-Based Approach*. Springer.

Manku, G. S. and Motwani, R. 2002. Approximate frequency counts over data streams. 346–357.

Marquezan, C. C., Wessling, F., Metzger, A., Pohl, K., Woods, C., and Wallbom, K. 2014. Towards exploiting the full adaptation potential of cloud applications. In *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*. PESOS 2014. 48–57.

Mell, P. and Grance, T. 2011. The nist definition of cloud computing. Tech. rep., National Institute of Standards and Technology (NIST).

Metsch, T. and Edmonds, A. 2011. Open cloud computing interface - infrastructure. *OCCI-WG*.

Michlmayr, A., Rosenberg, F., Leitner, P., and Dustdar, S. 2010. End-to-end support for qos-aware service selection, binding, and mediation in vresco. *IEEE T. Services Computing 3,* 3, 193–205.

Mok, A. K. and Liu, G. 1997. Efficient run-time monitoring of timing constraints. *Real-Time and Embedded Technology and Applications Symposium, IEEE 0*, 252.

Moscato, F., Aversa, R., Martino, B. D., Fortis, T.-F., and Munteanu, V. I. 2011. An analysis of mosaic ontology for cloud resources annotation. In *FedCSIS*. 973–980.

Moser, O., Rosenberg, F., and Dustdar, S. 2008. Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In *WWW*. ACM, 815–824.

OASIS-BPEL. 2007. Web Services Business Process Execution Language Version 2.0. Specification, Organization for the Advancement of Structured Information Standards (OASIS).

OWL. 2012. Owl 2 web ontology language document overview. Tech. rep.

Papadogiannakis, A., Vasiliadis, G., Antoniades, D., Polychronakis, M., and Markatos, E. P. 2012. Improving the performance of passive network monitoring applications with memory locality enhancements. *Computer Communications 35,* 1, 129–140.

Papazoglou, M. P. 2008. *Web Services: Principles and Technology*. Pearson, Prentice Hall.

Patnaik, D., Ramakrishnan, N., Laxman, S., and Chandramouli, B. 2012. Streaming algorithms for pattern discovery over dynamically changing event sequences. *CoRR abs/1205.4477*.

Popescu, R., Staikopoulos, A., Brogi, A., 0011, P. L., and Clarke, S. 2012. A formalized, taxonomy-driven approach to cross-layer application adaptation. *TAAS 7,* 1, 7.

Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., and Clarke, S. 2010. Taxonomy-driven Adaptation of Multi-Layer Applications using Templates. In *SASO*.

Richardson, I. and Lane, S. 2009. CD-JRA-1.1.4, Coordinated design knowledge models for software engineering and service-based computing . Tech. rep., S-cube. August.

Römer, K. 2006. Distributed mining of spatio-temporal event patterns in sensor networks. In *Euro-American Workshop on Middleware for Sensor Networks in conjunction with DCOSS 2006*. San Francisco, USA, 103–116.

Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S. O., Lorenzo, J., Mamelli, A., and Scholz, U. 2009. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science Series, vol. 5525. Springer, 164–182.

Römer, K. 2008. Discovery of frequent distributed event patterns in sensor networks. In *EWSN* (2008-01-27). Lecture Notes in Computer Science Series, vol. 4913. Springer, 106–124.

Sakurai, S., Ueno, K., and Orihara, R. 2008. Discovery of time series event patterns based on time constraints from textual data. In *Information and Mathematical Sciences*. Studies in Computational Intelligence.

Schmieders, E., Micsik, A., Oriol, M., Mahbub, K., and Kazhamiakin, R. 2011. Combining sla prediction and cross layer adaptation for preventing sla violations. In *2nd Workshop on Software Services: Cloud Computing and Applications based on Software Services.*

ShaikhAli, A., Rana, O. F., Al-Ali, R., and Walker, D. W. 2003. Uddie: An extended registry for web services. In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*. IEEE Computer Society.

Shanahan, M. 1999. The Event Calculus Explained. *Artificial intelligence today*, 409–430.

Sim, A. T. H., Indrawan, M., Zutshi, S., and Srinivasan, B. 2010. Logic-based pattern discovery. *IEEE Trans. Knowl. Data Eng. 22,* 6, 798–811.

Song, H., Raj, A., Hajebi, S., Clarke, A., and Clarke, S. 2013. Model-based cross-layer monitoring and adaptation of multilayer systems. *Science China Information Sciences 56,* 8, 1–15.

Spanoudakis, G. and Mahbub, K. 2006. Non-Intrusive Monitoring of Service-Based Systems. *International Journal of Cooperative Information Systems 15,* 3, 325–358.

Taher, Y., Boubeta-Puig, J., van den Heuvel, W.-J., Ortiz, G., and Medina-Bulo, I. 2013. A model-driven approach for web service adaptation using complex event processing. In *ESOCC Workshops*. Communications in Computer and Information Science Series, vol. 393. Springer, 346–359.

Tosic, V., Pagurek, B., Esfandiari, B., Patel, K., and Ma, W. 2002. Web service offerings language (wsol) and web service composition management (wscm). In *In Proc. of the Object- Oriented Web Services Workshop at OOPSLA 2002.*

Treiber, M. 2009. Models and mechanisms for coordinated service compositions. *S-Cube Deliverable CD-JRA-2.2.2.*

Villari, M., Brandic, I., and Tusa, F. 2012. *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*. IGI Global.

VMware. 2011. Timekeeping in vmware virtual machines. Tech. rep. December.

Vouk, M. A. 2008. Cloud computing - issues, research and implementations. *CIT 16,* 4, 235–246.

Webb, G. I. and Zhang, S. 2005. K-optimal rule discovery. *Data Mining and Knowledge Discovery 10.*

Wei, Y. and Blake, M. B. 2010. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing 14,* 6, 72–75.

Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., and Leymann, F. 2009. Monitoring and analyzing influential factors of business process performance. In *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International.* 141 –150.

White, S. 2004. Introduction to bpmn. Tech. rep.

Yousif, T. A. 2013. Application of thom's thermal discomfort index in khartoum state, sudan. *Journal of forest products and industries 2,* 5, 36–38.

Zaki, M. J. and Hsiao, C.-J. 2002. Charm: An efficient algorithm for closed itemset mining. In *SDM*. SIAM.

Zeginis, C. 2009. Monitoring the QoS of Web Services using SLAs - Computing metrics for composed services. M.S. thesis, University of Crete, Greece.

Zeginis, C., Konsolaki, K., Kritikos, K., and Plexousakis, D. 2011. "ECMAF: An Event-Based Cross-Layer Service Monitoring and Adaptation Framework". In *NFPSLAM-SOC*. Springer.

Zeginis, C., Konsolaki, K., Kritikos, K., and Plexousakis, D. 2012a. Ecmaf: An event-based cross-layer service monitoring and adaptation framework. In *Proceedings of the 2011 International Conference on Service-Oriented Computing*. ICSOC'11. Springer-Verlag, Berlin, Heidelberg, 147–161.

Zeginis, C., Konsolaki, K., Kritikos, K., and Plexousakis, D. 2012b. Towards proactive cross-layer service adaptation. In *WISE*. Vol. 7651. Springer, 704–711.

Zeginis, C., Kritikos, K., Garefalakis, P., Konsolaki, K., Magoutis, K., and Plexousakis, D. 2013a. Towards cross-layer monitoring of multi-cloud service-based applications. In *ESOCC*. 188–195.

Zeginis, C., Kritikos, K., and Plexousakis, D. 2014a. Event pattern discovery for cross-layer adaptation of multi-cloud applications. In *Service-Oriented and Cloud Computing*. Lecture Notes in Computer Science Series, vol. 8745. Springer Berlin Heidelberg, 138–147.

Zeginis, C., Kritikos, K., and Plexousakis, D. 2014b. *Event Pattern Discovery of Multi-cloud Service-based Applications.* Advances in Systems Analysis, Software Engineering, and High Performance Computing. IGI-global.

Zeginis, C. and Plexousakis, D. 2010a. Towards realizing ws cross-layer monitoring and adaptation. In *PhD Symposium of ECOWS'10*.

Zeginis, C. and Plexousakis, D. 2010b. Web service adaptation : State of the art and research challenges. Tech. rep., FORTH.

Zeginis, D., D'Andria, F., Bocconi, S., Gorronogoitia Cruz, J., Collell Martin, O., Gouvas, P., Ledakis, G., and Tarabanis, K. 2013b. A user-centric multi-PaaS application management solution for hybrid Multi-Cloud scenarios. *Scalable Computing: Practice and Experience 14,* 1, 17–32.

Zengin, A., Marconi, A., and Pistore, M. 2011. CLAM: Cross-layer Adaptation Manager for Service-Based Applications. In *QASBA '11*. ACM, 21–27.

Zhou, C., Chia, L. T., Silverajan, B., and Lee, B. S. 2003. UX—An Architecture Providing QoS-Aware and Federated Support for UDDI. In *the International Conference on Web Services (ICWS03)*.

Appendices

Appendix A

Event Meta-Model Description

Table A.1: Event meta-model: classes and properties analysis

Class: description	**Property**	**Description**	**Relationship with other classes / class properties**
Event: A superclass of every monitored event	*eventID*	the unique identifier of a monitored event	–

CompositeEvent: A complex event comprising two or more Events	*ordering*	Identifies the ordering of the two involved events (sequential or parallel)	–
	logicalOp	The logical operator (*AND*, *OR*) combining the two involved events	–
	timeInterval	The maximum time interval in millisecond for waiting for the second event to appear	–
	firstEvent	The first of the two events comprising a composite event	Event class
	secondEvent	The first of the two events comprising a composite event	Event class
SimpleEvent: A simple monitored event (either raw or aggregate)	*name*	The name of the metric of the event (e.g. average execution time)	–
	timestamp	The unix time of the event occurrence	–
	criticality	The criticality (success, warning or critical) of the monitored event determined after assessement	–
	hasLayer	Defines the layer of the monitored event, either a Cloud or a SOA one	Layer class
	hasComponent	The source component that produced the event (they defined in the respective component model)	Component class
Layer: The SOA or Cloud layer, where the monitored event was detected	*name*	The name of the layer, i.e., IaaS, PaaS, BPM or SCC	–
Component class: The source component of the respective application's component model that emitted the event	*componentID*	The unique identifier of the source component (e.g. 7001), as it is defined in the component model	componentID property of the application's component model

	componentName	The name of the source component (e.g. FlexiantVM_CPU) as it is defined in the component model	componentName property of the application's component model
	state	The state of the component, i.e., active or inactive	type property of the application's component model
	type	The type of the source component defines its high-level Cloud layer, i.e., IaaS, PaaS or SaaS	IaaS_component, PaaS_component or SaaS_component classes of the application's component model
Functional_event: An event related to functional characteristics of the application	*type*	The type of the functional event, as it is described by the class' subclasses (e.g. softwareEvent, I/O_event, HardwareEvent, etc.)	–
NonFunctional_event: An event related to non-functional/performance characteristics of the application (e.g. KPI-violation, SLA-violation or ContextModification)	*propertyName*	The name of the related property/metric (e.g. throughput, availability, etc.)	–
	propertyURI	The OWL_Q URI defining the property/metric definition	–
	operator	The operator that should be applied on the assessment of the event	–

	constraintURI	The WSLA URI defining the SLO and its assessment function	–
NumericNF_event: A numeric non-functional event (e.g. service throughput, execution time, etc.)	*detectedNumericValue*	The value measured by the corresponding monitoring mechanism	subclass of NonFunctional_event class
	warningSLO	The warning threshold of the SLO	
	criticalSLO	The critical threshold of the SLO	
EventPattern: An event pattern discovered mapping to an association rule	*patternID*	The unique identifier of the event pattern	–
	eventNumber	The number of events comprising the event pattern	–
	frequency	The number of occurrences of the event pattern in a specific event stream	–
	causing_event	The causing event (either simple or composite) leading to a specific SLO violation	Event class
	caused_event	The caused event (i.e. SLO violation) of the discovered event pattern	SimpleEvent class
AdaptationRule: The adaptation rule mapping to the discovered event pattern	*ruleID*	The unique identifier of the adaptation rule	–
	name	The name of the adaptation rule (i.e. an if...then clause)	–
	hasEventPattern	The property mapping the adaptation rule with the corresponding event pattern	SimpleEvent class
	fireActions	The property mapping the adaptation rule with the corresponding fired adaptation actions	–
AdaptationAction: The adaptation action dictated by the corresponding adaptation rule	*actionID*	The unique identifier of the adaptation action	–
	name	The name of the adaptation action (e.g. FlexiantVM scaling up)	–

Table A.1: Event meta-model: classes and properties analysis

Appendix B

Component Meta-Model Description

Table B.1: Component meta-model: classes and properties analysis

Class: description	**Property**	**Description**	**Relationship with other classes / class properties**
Component: A superclass of every component producing events	*componentID*	the unique identifier of a component	–

	componentName	The name of the component (e.g. FlexiantVM CPU)	–
	state	The state of the component (i.e. active or inactive)	–
IaaS_component: An IaaS layer component	–	–	subclass of InfrastructureComponent class
AssistantDevice: An assistant device used by a service or software (e.g. temperature sensor)	–	–	subclass of InfrastructureComponent class
InfrastructureComponent: A super class at the infrastructure layer	–	–	subclass of Component class
VM: A virtual machine hosting an application part	*OS*	The operating system running on the VM	–
	OS_version	The version of the operating system(e.g. Ubuntu 12.04)	–
	OS_latestVersion	The latest version of the operating system(e.g. Ubuntu 14.04)	–
	provider	The provider of the VM (e.g. Flexiant)	–
	datacenter	The VM's datacenter	–
	cluster	The cluster of the VM (e.g. "low", "medium", "high")	–
	location	The country location of the VM (e.g. the UK)	–

	includesCompute	The compute node of the VM	Compute class
	includesStorage	The storage node of the VM	Storage class
	includesNetwork	The network node of the VM	Network class
	hasVMSLA	Defines the SLA of the non-functional characteristics of the VM	SLA class
Compute: The compute node of a VM (i.e. CPU or memory)	–	–	contained class of VM class
CPU: The CPU of the hosting VM	*cores*	The number of the CPU's cores (e.g. 2 or 4)	subclass of compute class
	frequency	The frequency of the CPU (e.g. 2,4GHz)	
	architecture	The architecture of the CPU (i.e. 32bit or 64bit)	
	upgradeable	A boolean value defining the if the CPU can be upgraded based on the motherboard's chipsets interface	
Memory: The RAM memory of the hosting VM	*amount*	The total amount of memory (e.g. 8GB)	subclass of compute class
	maxMemory	The max RAM memory supported by the motherboard	
	frequency	The frequency of the RAM memory (e.g. 1333MHz)	
	availableSlots	The number of motherboard's available RAM slots	
Storage: The storage disk of the hosting VM	*capacity*	The capacity of the disk (e.g. 1TB)	containing class of VM class
	type	The type of the disk (i.e. HDD or SSD)	

	speed	The rpm speed of the disk (e.g. 7200rpm)	
Network: The network of the hosting VM	*bandwidth*	The internet connection bandwidth (e.g. 10Mbit/s)	containing class of VM class
	wirelessProtocol	The security protocol of the wireless network (e.g. WPA2, WEP)	
	ISP	The ISP of the network	
	availableISPs	All the available ISPs	
router: The router transmitting the network's bandwidth	*routerInterface*	The router's interface (e.g. ethernet, token ring))	subclass of network class
	firmwareVersion	The version of the router's firmware	
	firmwareLatestVersion	The latest version of the router's firmware	
Switch: The switch used in the network	*ports*	The number of switch's ports	subclass of network class
Sensor: A sensor utilized by a service or software	*batteryLevel*	The sensor's battery percentage level	subclass of the AssistantDevice class
	frequency	The current transmitting frequency of the measured values	
	maxFrequency	The max transmitting frequency of the measured values supported by the sensor	
	type	The sensor's type (e.g. temperature sensor)	
	removableBattery	Defines a removable or a built-in battery	
	rechargeable	Defines the rechargeability of the battery	

MobileDevice: A mobile device utilized by a service or software	*OS*	The operating system of the mobile device	subclass of the AssistantDevice class
	OS_version	The current version of the mobile device’s OS	
	OS_latestVersion	The latest version of the mobile device’s OS	
CellPhone: A cell phone utilized by a service or software for alerting purposes	*supported network*	The familiar network operator of the cell phone (e.g. cosmote, vodafone)	subclass of the MobileDevice class
	availableNetworks	The available network operators	
	maxMemory	The maximum memory supported by the cell phone (i.e. SD card storage)	
	memoryIntensiveApps	A list of the most memory intensive running applications	
	storageIntensiveApps	A list of the applications occupying high internal capacity	
GPS: A GPS device utilized by a service or software for alerting purposes	*current satellite*	The name of the satellite currently providing the geographical location	subclass of the MobileDevice class
	availableSatellites	The available satellites	
	mapsProvider	The provider of the GPS’s maps (e.g. Cygic, TomTom)	
	mapsVersion	The version of the installed maps	
	mapsLatestVersion	The latest version of the installed maps	
Tablet: A tablet device utilized by a service or software for alerting purposes	*memoryIntensiveApps*	A list of the most memory intensive running applications	
	storageIntensiveApps	A list of the applications occupying high internal capacity	
PaaS_component: A PaaS layer component (e.g. Datastore, Appserver)	–	–	Subclass of component class

Platform: A cloud platform providing a set of add-ons (i.e. PaaS components)	*provider*	The PaaS provider (e.g. Heroku, Windows Azure)	–
	URL	The platform's URI	–
	Description	A brief description of the platform	–
	verticalScaling	Defines the vertical scaling capabilities of the platform	–
	horizontalScaling	Defines the horizontal scaling capabilities of the platform	–
	providesComponent	Defines the PaaS components provided by the specific platform	–
SaaS_component: A SaaS layer component (e.g. a service, an SLA, etc.)	*runsOnPaaS*	The PaaS where the SaaS component runs on	PaaS_component class
	runsOnIaaS	The IaaS where the SaaS component runs on	IaaS_component class
BPM_Component: A BPM component (e.g. Business Process, KPI)	–	–	subclass of SaaS_component class
BusinessGoal: A functional goal of the business process	*includesBG*	Defines sub-goals of the business goal	contained class of BusinessGoal class
BusinessProcess: A business process defining the workflow of a SBA	*includesBusinessProcess*	Defines sub-processes of the SBA main business process	contained class of the BusinessProcess class
	includesControlFlowPattern	Defines the control flow patterns of the BP	ControlFlowPattern class

	includesDataFlow	Defines the data flows of the BP	DataFlow class
	includesTask	Defines tasks comprising the BP	Task class
	requiresIn	Defines the input required by the BP	Input class
	providesOut	Defines the output provided by the BP	Output class
	hasBG	Defines the business goal of the BP	BusinessGoal class
	hasKPI	Defines the KPIs of the SBA's business process	KPI class
BP_model: The model of the SBA's BP	*version*	Defines version of the BP model	–
	describesBP	Defines the BP described by the model	BusinessProcess class
KPI: A KPI defining non-functional constraints for the BP	*metric*	Defines the specific metric of the KPI	–
	operator	Defines the operator to be used during the KPI assessment	–
	threshold	Defines the threshold of the KPI constraint	–
	mapsToSLO	Defines SLO mapping to this KPI	–
Task: A single task/activity of the BP	*mapsToOperation*	Defines the operation realizing the specific task	Operation class
Role: The role performing the specific task/activity	*performsTask*	Defines the task realized by the specific role	Task class
	name	The name of the role	–
	availability	Defines the availability of the role	–

DataFlow: A data flow between two or more activities of the BP	*includesDataBinding*	Defines the binding of the data flow	containing class of the DataFlow class
DataBinding: The binding connecting an output of one service with the input of another service	*bindsToIn*	Defines the input of the data binding	Input class
	bindsToOut	Defines the output of the data binding	Output class
ControlFlowPattern: A control flow pattern between the BP's activities	*type*	Defines the type of the control flow pattern (e.g. sequential, parallel	–
	condition	The condition to be assessed for the pattern selection	–
	includesComponentPositioning	Defines the positioning of a task in the BP	–
	includesCF_positioning	Defines the positioning of the control flow in the BP	–
ComponentPositioning: The positioning of a component (i.e. task) in the BP	*position*	The exact position in the BP	–
	value	The assessed value to select this component positioning	–
	bindsToTask	Defines the positioning of the exact task in the BP	Task class
CF_positioning: The positioning of a control flow in the BP	*position*	The exact position in the control flow	containing class of the ControlFlowPattern class
	value	The assessed value to select this control flow positioning	–
Input: The input required by a service or a BP activity	*type*	The input type (e.g. integer, string)	–
	stringValue	The string value of a service or activity input	–
	numValue	The numeric value of a service or activity input	–

	mapsToIn	Defines the mapping of inputs between a BP activity and the corresponding service	Input class
Output: The output provided by a service or a BP activity	*type*	The output type (e.g. integer, string)	–
	stringValue	The string value of a service or activity output	–
	numValue	The numeric value of a service or activity output	–
	mapsToIn	Defines the mapping of an input and an output between a BP activity and the corresponding service	Input class
	mapsToOut	Defines the mapping of outputs between a BP activity and the corresponding service	Input class
Service: A web service realizing a BP activity	*URI*	The URI for accessing the web service	–
	WSDLbinding	The type of WSDL binding (i.e. Document, RPC)	–
	validBindingStyles	The valid binding styles of the service	–
	containsOperations	Defines the operations of the web service	Operation class
	requiresIn	Defines the input of the web service	Input class
	providesOut	Defines the output of the web service	–
	hasSLA	Defines the agreed SLA between the service provider and consumer	SLA class
	requiresAssistantDevice	Defines the assistant device required for the service’s functionality	AssistantDevice class

SimpleService: A simple web service	–	–	Subclass of Service class
CompositeService: A composite web service	*containsControlFlowPattern*	The control flow pattern linking the individual simple web services	ControlFlowPattern class
	containsDataFlow	The data flow linking the individual simple web services	DataFlow class
Operation: An individual operation of a web service realizing a BP activity	*requiresIn*	The input required by the operation	–
	providesOut	The output provided by the operation	–
SCC_component: An SCC component (e.g. service, operation)	–	–	subclass of SaaS_component class
Software: A software utilized by an SBA web service (e.g. CEP engine, rule engine)	*version*	The current version of the software	
	latestVersion	The latest stable version of the software	subclass of SaaS_component class
	configuration	The proper configuration of the software for the SBA execution	
	usesAssistantDevice	An assistant device required by the software (e.g. special sensor)	
Profile: The profile describing the functional and non-functional characteristics of a web service	*describesService*	Defines the service described by the profile	Service class

FunctionalProfile: The functional profile of a web service	–	–	subclass of Profile class
Conditions: The conditions realizing the functional profile	*object*	The object for which the condition holds	–
	operator	The operator of the condition (e.g. Number of inputs=2	–
	value	The value/threshold of the condition	–
	describesProfile	The functional profile of the conditions	FunctionaProfile class
Preconditions: The preconditions that must hold for the specific service	–	–	subclass of Conditions class
Postconditions: The postconditions that must hold for the specific service	–	–	subclass of Conditions class
NonFunctionalProfile: The non functional profile of a web service describing performance characteristics	–	–	subclass of Profile class
Constraint: Defines a constraint of the non functional profile	*describesNFprofile*	The non functional profile referred by the constraint	NonFunctionalProfile class
SimpleConstraint: A simple constraint mapping to an SLO	*metric*	The metric of the simple constraint	–
	threshold	The constraint's threshold	–

	operator	The constraint’s operator	–
	mapsToSlo	The SLO mapping to the constraint	SLO class
CompositeConstraint: A composite constraint containing other individual constraints	*logicalOperator*	The logical operator linking the individual constraints (e.g. AND, OR)	–
	containsConstraint	Defines the containing constraints of the composite constratint	–
SLA: The SLA contracted by the SBA provider and the consumer	*consumer*	The SBA’s consumer	–
	provider	The SBA’s provider	–
	date	The date of the SLA contract	–
	validity_period	The valdiity period of the SLA	–
	containsServiceLevel	The levels of an SLA	ServiceLevel class
ServiceLevel: The service levels of the contracted SLA	*requirements*	The requirements for the specified service level	–
	penalty	The posed penalty when the service level is violated	–
	level	The specified level number (1 is the strictest level)	–
	cost	The cost for serving this service level	–
SLO: An SLO of the specified SLA’s level	*metric*	The metric of the SLO	–
	threshold	The SLO’s threshold	–
	operator	The SLO’s operator	–