UNIVERSITY OF CRETE
SCHOOL OF SCIENCES AND ENGINEERING
COMPUTER SCIENCE DEPARTMENT
VOUTES UNIVERSITY CAMPUS, HERAKLION, GR-70013, GREECE

A Distributed Cross-Layer Monitoring System based on QoS Metrics Models

Damianos Metallidis

Thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

Thesis Advisor: Prof. *Dimitris Plexousakis*

October 2016

This work has been performed at **Information Systems** laboratory, **Institute of Computer Science (ICS), Foundation for Research and Technology – Hellas (FORTH)**, and is partially supported by the CloudSocket that has been funded within the European Commissions H2020 Program under contract number 644690.

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

A Distributed Cross-Layer Monitoring System based on QoS Metrics Models

Thesis submitted by
Damianos Metallidis
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Damianos Metallidis

Committee approvals: _____
Dimitris Plexousakis
Professor, Thesis Supervisor

_____
Evangelos Markatos
Professor, Committee Member

_____
Kostas Magoutis
Assistant Professor, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, October 2016

Abstract

In order to implement cross-organizational workflows and to realize collaborations between small and medium enterprises (SMEs) the use of Web service technology and Service-Oriented Architecture has become a necessity. Whilst, SMEs are continuously moving towards service-oriented infrastructures where applications are being modeled, the need of hosting them has raised an important issue for the quality of the underlying cloud infrastructures. Virtualization, offered by cloud infrastructures, delegates the use of any kind of resources, such as computing environments or storage systems, to the data center's internal networks. All of the above issues raised the need for monitoring of the quality of the acquired resources and of the services offered to final users as also the workflow-based procedures used by SMEs in order to use services. Although, most of the monitoring frameworks and systems rely only on a layer-specific quality model, covering, e.g., the service layer, without considering the cross-layer dependencies it might have with the Quality Models (QMs) of Workflow and Infrastructure.

To alleviate this problem in this thesis, we have defined three metric QMs that cover quality terms based on Workflow, Service and Infrastructure layers. These quality terms could refer to *(a)* quality dimensions describing the quality aspect which can be used to provide an aspect-specific partition of quality terms *(b)* quality attributes indicating properties of an object (e.g., object of response time) *(c)* raw quality metrics representing raw information that could be taken from monitoring sensors and *(d)* composite metrics which can be used in the calculation of different composite metrics by applying a specific metric formula which is defined in order to calculate the value of the according composite metric. The novelty of our approach is the definition of a fourth one QM depicting two types of cross-layer dependencies, between the quality metrics of the aforementioned three QMs, which could be *(1)* equality relations and *(2)* inter-dependency relations. Furthermore, in order to compute the values of the quality metrics for each of these three QMs we have defined aggregators, computation formulas, and algorithms which are implemented and placed within a distributed cross-layer monitoring system. Implementation of the cross-layer monitoring system is based upon the two open-source monitoring tools of Prometheus and Nagios, for Service and Infrastructure layers accordingly. As a workflow engine we have used the Activiti open source project, responsible for the execution and monitoring of cross-organizational workflows.

We evaluated the proposed cross-layer monitoring system by deploying it's components in the private cloud of VMWare and in the public cloud provider of University of Crete Data Center (UCDC). Our evaluation procedure is based on *(a)* the exposition of a real life workflow example indicating the virtue of the unified dependency quality model proposed and explicates the way computation algorithms work *(b)* the performance aspect of the proposed cross-layer monitoring system such as the query execution latency upon the management databases being used, in both private and public cloud providers and *(c)* the value of accuracy offered by the distributed monitoring system.

Περίληψη

Η χρήση της τεχνολογίας των υπηρεσιών Διαδικτύου και της αρχιτεκτονικής βασισμένης σε υπηρεσίες για την εφαρμογή δια-οργανωτικών ροών εργασίας και την σύμπραξη μεταξύ των μικρών και μεσαίων επιχειρήσεων (ΜΜΕ), έχει καταστεί απαραίτητη. Ενώ, οι ΜΜΕ συνεχώς κινούνται προς την κατεύθυνση των υπηρεσιακών υποδομών όπου αναπτύσσονται πρότυπα εφαρμογών, η ανάγκη για την φιλοξενία των εφαρμογών έθεσε ένα σημαντικό ζήτημα για την ποιότητα των υποκείμενων υποδομών νέφους. Οι τεχνικές της εικονοποίησης (virtualization), που προσφέρονται από τις υποδομές νέφους, αντιπροσωπεύουν την χρήση οποιουδήποτε είδους πόρων, όπως υπολογιστικά περιβάλλοντα ή συστήματα αποθήκευσης, σε εσωτερικά δίκτυα του εκάστοτε κέντρου δεδομένων. Όλα τα παραπάνω θέματα που τέθηκαν φέρουν την ανάγκη για την παρακολούθηση της ποιότητας των αποκτηθέντων πόρων και των υπηρεσιών που προσφέρονται στους τελικούς χρήστες, όπως επίσης και οι διαδικασίες των ροών εργασίας που χρησιμοποιούνται από τις ΜΜΕ, προκειμένου να χρησιμοποιηθούν αυτές οι υπηρεσίες. Ωστόσο, τα περισσότερα από τα πλαίσια και τα συστήματα παρακολούθησης βασίζονται μόνο σε ένα ειδικό μοντέλο ποιότητας, που καλύπτει, π.χ., το στρώμα υπηρεσίας, χωρίς να λάβει υπόψη τις εξαρτήσεις επιπέδων που θα μπορούσε να έχει με τα μοντέλα ποιότητας των ροών εργασίας και υποδομών .

Για να αντιμετωπιστεί αυτό το πρόβλημα σε αυτή την εργασία, έχουμε ορίσει τρία μετρικά μοντέλα ποιότητας που καλύπτουν τους όρους της ποιότητας βασιζόμενα στα στρώματα των ροών εργασίας, υπηρεσιών και υποδομών. Οι ποιοτικοί αυτοί όροι αναφέρονται (α) στις διαστάσεις ποιότητας, που περιγράφουν την πτυχή της ποιότητας που μπορεί να χρησιμοποιηθεί για να παρέχει μια ειδική κατάτμηση των όρων ποιότητας (β) στις ποιοτικές ιδιότητες οπού αναφέρονται στις ιδιότητες ενός αντικειμένου (π.χ., αντικείμενο του χρόνου απόκρισης) (γ) στις μετρικές ποιότητας που αντιπροσωπεύουν πληροφορίες που θα μπορούσαν να ληφθούν από τους αισθητήρες παρακολούθησης και (δ) στις σύνθετες μετρικές ποιότητας που μπορούν να χρησιμοποιηθούν για τον υπολογισμό των διαφόρων σύνθετων μετρικών ποιότητας εφαρμόζοντας μια συγκεκριμένη μετρική φόρμουλα η οποία ορίζεται με σκοπό να υπολογιστεί η τιμή της αντίστοιχης σύνθετης μετρικής μεταβλητής. Η καινοτομία της προσέγγισής μας είναι ο ορισμός ενός τέταρτου μοντέλου ποιότητας όπου απεικονίζει δύο τύπους εξαρτήσεων μεταξύ των ποιοτικών μετρήσεων των προαναφερθέντων τριών μοντέλων ποιότητας, οι οποίοι είναι πιθανό να είναι (1) σχέσεις ισότητας και (2) σχέσεις αλληλεξάρτησης. Επιπλέον, προκειμένου να υπολογιστούν οι τιμές των ποιοτικών μετρικών για κάθε ένα από τα τρία αυτά ποιοτικά μοντέλα έχουμε ορίσει ειδικούς συλλέκτες, τύπους υπολογισμών, και αλγόριθμους όπου εφαρμόζονται και τοποθετούνται μέσα σε ένα κατανεμημένο σύστημα παρακολούθησης πολλαπλής στιβάδας. Η εφαρμογή του πολυεπίπεδου συστήματος παρακολούθησης βασίζεται σε δύο εργαλεία παρακολούθησης ανοιχτού κώδικα, του Prometheus και του Nagios, για τα στρώματα των υπηρεσιών και των υποδομών αναλόγως. Ως μηχανή ροής εργασίας έχουμε χρησιμοποιήσει το λογισμικό ανοικτού πηγαίου κώδικα Activiti, υπεύθυνο για την εκτέλεση και παρακολούθηση των δια-οργανωτικών ροών εργασίας.

Αξιολογήσαμε το προτεινόμενο πολυεπίπεδο σύστημα παρακολούθησης παρατάσ-

σοντας τις εκάστοτε συνιστώσες του στις υπηρεσίες ιδιωτικού νέφους της VMWare και στο δημόσιο πάροχο νέφους του Πανεπιστημίου Κρήτης και Κέντρο Δεδομένων (UCDC).Η διαδικασία αξιολόγησής μας βασίζεται (α) στην έκθεση ενός παραδείγματος ροών εργασίας που δείχνει την αρετή του ενιαίου μοντέλου ποιότητας καθώς και στην χρήση των αλγορίθμων υπολογισμού,(β) στην πτυχή της απόδοσης του προτεινόμενου πολυεπίπεδου συστήματος παρακολούθησης, όπως είναι ο υπολογισμός της καθυστέρησης της εκτέλεσης του ερωτήματος από τις βάσεις δεδομένων διαχείρισης που χρησιμοποιούνται, τόσο στον ιδιωτικό όσο και στον δημόσιο πάροχο νέφους και (γ) η αξία της ακρίβειας η οποία προσφέρεται απο το κατανεμημένο σύστημα παρακολούθησης.

Acknowledgements

First of all, i would like to thank my advisor, Professor Dimitris Plexousakis for his important guidance throughout this thesis and for giving me the chance to work with his team. Furthermore, i would like to thank the rest of the committee, Kostas Magoutis and Evangelos Markatos for the valuable comments and the interesting questions during my defense.

Next, i would like to thank the members of Institute of Computer Science (ICS), Foundation of Research and Technology Hellas (FORTH) and more specifically from the lab of Information System Laboratory (ISL). I would like to thank Kyriakos Kritikos for showing me the right direction of my thesis main theme and also my special thanks goes to Chrysostomos Zeginis who has been a major supporter during the development of the thesis as the results would not be same without Chrysostomos comments and directions.

Furthermore, i would like to thank my colleague and friend Georgios Chatzipantelis for his continues phycological support during the thesis development. My thanks, also goes to my friends from Thessaloniki, Georgios Stamoulis and Nikos Mwysiadis who where a great company during this period.

Lastly, i would like to thank my parents, Olga and Giorgos and my brother Kostas for their endless love, support and encouragement through my studies and my career decisions, as they always believed in me.

This work has been performed at **Information Systems** laboratory, **Institute of Computer Science (ICS), Foundation for Research and Technology – Hellas (FORTH)**, and is partially supported by the CloudSocket that has been funded within the European Commissions H2020 Program under contract number 644690.

An early report of this work appeared in the proceedings of the 1st International Workshop on Performance and Conformance of Workflow Engines, co-located with the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC 2016), Vienna, Austria, September 5th, 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement and proposed solution

Nowadays, many organizations form dynamic co-operations in order to effectively deal with market requirements. Companies focus on their core business and goals whilst outsourcing secondary activities to other organizations. Growing complexity of products requires co-makership relations between organizations. Value chains require a tight cooperation between companies participating in these chains. To enable co-operation between oraganizations, the information processing infrastructures of the participating small and medium enterprises (SMEs) need to be linked. Workflow management systems that control the processes in the individual SMEs are a key feature. In today's businesses, the application of workflow management systems (WFMSs) for automated process support is widespread. Traditionally, the emphasis of workflow management is based within the boundaries that organization themselves were possessed to. Using workflow support in virtual organizations, however, implies the extension of the functionality of it, such that workflow management systems in *different organizations* can be linked to manage integrated *cross-organizational workflows*. The extended workflow support must be able to (a) deal effectively with differently workflow environments, and (b) dynamically formate new and current cross-organizational workflows. Thus, for example a cross-organizational workflow should be able to allow one organization (the service consumer) to start a process (a service or a scientific experiment) on its behalf in another organization (the service provider) and receive the results of this process. As blackbox processes are too unrefined for tightly partnership organizations, advanced monitoring and control mechanisms are required to support fine-grained interaction between these organizations, that deal with cross-organizational workflows, whilst also preserving their autonomy to a great extent.

In order to implement cross-organisational workflows and to realise collaborations between small and medium enterprises, the use of Web service technology and Service-oriented Architecture has become a necessity. Whilst, SMEs are continuously moving towards service-oriented infrastructures where applications are being modeled, the need of hosting them has raised an important issue for the quality of the underlying cloud infrastructures. Virtualization, that cloud infrastructures provide, delegates the use of any

kind of resources, such as computing environments or storage systems, to the data center's internal networks. All of the above issues raised the need for monitoring of the quality of the acquired resources and of the services offered to final users as also the workfload-based procedures used by SMEs in order to use services.

Workflow, Service and Infrastructure monitoring phases concerns themselves with specific level measurements. Monitoring for those three functional layers is the continuous and closed loop procedure of measuring, monitoring, reporting, and improving the QoS of systems and applications delivered by service-oriented solutions. Monitoring involves several distinct activities including logging and analysis of workflow/service/infrastructure execution details, obtaining metrics for each of the three functional layers, detecting business situations that require management attention, determining appropriate control actions to take in response to business situations,and using historical service performance data for continuous improvement and update in the stages of workflow,service and infrastructure.

QoS refers to the ability of the Web service to respond to expected invocations and to perform them at the commensurate level with the mutual expectations of both its provider and its customers. Quality factors that reflect customer expectations, such as constant service reliability, security, fidelity, availability, connectivity, scalability and elasticity, become key to keeping a business competitive and viable as they can have a serious impact upon service provision. Thus, the quality of workflows,services and of the underlying infrastructure becomes an important criterion that determines the usability and utility,which influence the popularity of a particular Web service, and an important selling and differentiating point between Web services providers.

We address the aforementioned quality aspects by specifying Quality Models (QMs) that a monitoring framework could adopt so as to gather monitoring data taken from three different layers: (a) *Workflow*, (b) *Service* and (c) *Infrastructure* layer. State-of-the-art research is based mostly on individual layers without considering the cross-layer dependencies [1] that Quality Models (QM) might have. This leads us to propose individual-layer-specific QMs along with a glue QM catering for cross-layer dependencies whenever applicable. Intention of QMs is the specification of quality terms (e.g., quality attributes) as well as of the respective relationships between these terms. In order to specify the legitimate structure of a QM, the respective conceptualisations and all possible types of quality term/concept relationships, a Quality Meta-Model (QMM) [2] should be in place. While different QMMs have been proposed using different representation formalisms, ontologies seem to be the best formalism as they provide a formal, syntactic and semantic description model of concepts, properties and dependencies between concepts. Moreover, they are extensible, human-understandable and machine-interpretable and enable reasoning via using Semantic Web technologies.

Catering for the workflow based usage of a service as well as the infrastructure that supports software components realizing part of the workflow functionality, we take in account three main QMs: (a) the Workflow QM (*WM*) stressing quality terms related to tasks and workflows, (b) the Service QM (*SM*) indicating quality terms for assessing the quality of services, and (c) the Infrastructure QM (*IM*) encompassing quality terms suitable for the assessment of the quality of the underlying cloud infrastructure. Additionally, we have

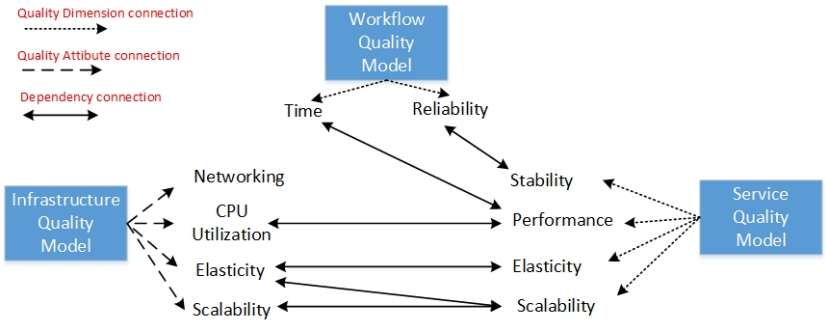


Figure 1.1: Quality Models terms and cross-layer dependencies

defined cross-layer relationships between quality metrics defined in the three QMs, in a sense that a metric defined in layer *X* can be used for the calculation of a metric defined in layer *Y*. Figure 1.1 depicts the quality models and terms that have been defined along with the cross-layer connection dependencies between them, indicating relations between quality metrics.

We are heavily focused on WM quality metrics, in a manner such that a possible user of any complicated workflow is able to evaluate results of procedures that include benchmarking and conformance tests based on the aforementioned aspects. Following this direction we will be able to deliver Workflow *Monitor* as a Service (WMaaS) in any Workflow as a Service (WfaaS) procedure. The definitions of the quality terms that are being proposed, as well as the provision of the distributed monitoring solution, will help SMEs to give performance and reliability grades on their business workflows, standardizing the way that the workflows are going to be evaluated.

1.2 Thesis Organization

The rest of the thesis is structured as follows. In Chapter 2 we review the related work, while in Chapter 3 we represent the four Quality Models. Chapter 4 demonstrates the architecture of the monitoring system along with the logical and technical challenges that we have faced. Chapter 5 describes the implementation procedure of the monitoring system and Chapter 6 provide the details about the implementation and functionality of the quality metric aggregators. We validate the proposed distributed monitoring system in Chapter 7 of Evaluation with various experiments and in Chapter 8 we conclude and depict directions of ongoing and future work.

Chapter 2

Related Work

2.1 Monitoring Approaches

This section reviews a number of research and industrial approaches dealing with monitoring systems and the according Quality Models that are being composed of. We also make a comparison of the proposed research solutions along with the monitoring solution offered in the current thesis, in order to indicate the drawbacks and advantages of each of one them.

2.1.1 Quality Models and Monitoring Solutions

Monitoring and adaptation for service-based systems, cross-layer adaptation and monitoring for service-based applications [3] have been researched as precursors for advanced systems for failure prevention and recovery in the cloud. Based on [3] Cross-layer adaptation and monitoring (CLAM) is an approach to the run-time quality assurance of service-based applications (SBAs). An analytical approach to an existing platform-as-a-service framework, revealed the different third party services and their characteristics, as a precursor to defining SLAs. Aim of the CLAM is to monitor the different layers of an SBA and correlate the monitoring results, such that in the event that a problem occurs an effective adaptation strategy is inferred for enacting a coordinated adaptation across all layers of the SBA. More precisely they have provided three layers

- The top layer of an SBA is the business process management (BPM) layer and it concerns the business level aspects of an SBA, such as process workflows, service networks, key performance indicators, and process performance metrics. The BPM layer focuses mostly on monitoring business activities and manages the performance of the business.

- The middle layer of an SBA is the service composition and coordination layer (SCC), which concerns the composition of individual services into new services, the functional (e.g. service behaviour) and non-functional quality of service (QoS) (e.g. responsiveness and availability) characteristics of the individual services or

the composed services. The SCC layer focuses mostly on both run-time verification and testing of the service behaviour, and monitoring the QoS of the individual or the composed services.

- and lastly in the bottom layer of an SBA is the service infrastructure (SI) layer and concerns the software (e.g. service middleware, service registry) and the hardware (e.g. compute, storage, bandwidth) resources utilised in an SBA

Although, in this approach researchers focused on the definition of SLAs in the BPM, SCC, and SI layers, for cross-layer adaptation and monitoring of SBAs, they were not as analytical as to capture and analyze quality terms of fidelity and availability in each of the layers that have been proposed. Also the layer of Workflow monitoring was not considered which is valuable in order to characterize quality aspects of the BPM layer.

The extensive state-of-the-art and research baseline survey [4] identified challenges to be addressed, which include (i) what data should be collected and what metrics used; (ii) how brokers should manage large volumes of events collected from heterogeneous sources; and (iii) what kinds of analysis and prediction techniques should be used to support proactive failure prevention in order to enable continuous quality assurance. In [5] researchers presents the design, implementation and validation of a novel run-time monitoring architecture for conversational services, which aims to provide a monitoring framework enabling the integration of different verification tools and higly based on the procedure that messages are being exchanged between services and monitoring managers. More precisely three different approaches for message interception have been proposed:

- First a Handler-Based Interception which is attached to the monitored service. The request/response messages are forwarded first to the handler, thus a handler is able to intercept them before reaching the monitored service and the consumer respectively.

- Second a wrapper-based Interception where the monitored service is wrapped within another service. The resulting service has the same interface as the monitored service, and it delegates the messages to the monitored service. Thus, it is able to intercept the request/response messages exchanged between the monitored service and the consumer.

- and third a Proxy-based Interception where an intermediate node acts as a network proxy. The proxy is able to intercept the request/response messages passing over the transport protocol, before they reach their destination.

The architecture has been designed and implemented to facilitate integration with the existing service-oriented architectures, and to allow the use of different monitoring approaches based on the technique of message interception. Our work from the other hand, has tackled this aspect with the usage of a publish/subscribe system and placed the monitoring functionality upon different components making these way a clear separation between messages being exchanged and the manner that they are being leveraged.

In [6] researchers developed and validate CLAMBS—Cross-Layer MultiCloud Application Monitoring and Benchmarking as-a-Service for efficient QoS monitoring and benchmarking of cloud applications hosted on multi-clouds environments. Advantage of CLAMBS is its capability of monitoring and benchmarking individual application components such as databases and web servers, distributed across cloud layers (*-aaS), spread among multiple cloud providers. Using experimentation and prototype implementation, CLAMBS is flexible and resource efficient and can be used to monitor several applications and cloud resources distributed across multiple clouds. CLAMBS is based upon three main components:

- The CLAMBS Manager which is a software component that performs two operations: 1) it collects QoS information from Monitoring Agents; and 2) it also collects benchmarking information from benchmarking agents running on several virtual machines (VMs) across multi-cloud providers and environments. In case of monitoring, the manager collects QoS parameter values from the monitoring agents running at the *aaS layers. The communication between the manager and the agents can employ a push or pull technique. In case of pull technique, the manager polls the CLAMBS monitoring agents at different frequency to collect and store the QoS statistics in a local database (DB).

- Another major component of the CLAMBS framework is the monitoring agent. The monitoring agent resides in the VM running the application and collects and sends QoS values as requested by the manager. After the monitoring system initialization, the agent waits for the incoming requests from the manager or starts to push QoS data to the manager. Upon arrival of the request, the agent retrieves the stated QoS values belonging to a given application process and/or a system resource and sends them back as a response to the manager.

- and finally the third component of the CLAMBS framework is the benchmarking agent. This agent has the capability to migrate from the manager VM to a VM that either hosts the application/service or act as a client to the service. The benchmarking agent incorporates standard functions to measure the network performance between the data center(s) hosting the application service and the client and also incorporates a load-generating component that generates traffic to benchmark the application based on a workload model.

Nevertheless, CLAMBS is not stating the possible dependencies between quality metrics of the different functional layers that it supports such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), and also does not take into account the layer of Workflow processing.

In [7], a novel monitoring architecture is addressed that deals with the aspect of cloud provider and cloud consumer. This architecture offers a monitoring PaaS to each cloud consumer that allows to customize the monitoring metrics. This is accomplished by the means of an adaptive distributed monitoring architecture automatically deployed in the cloud infrastructure. An intensive empirical evaluation of performance and scalabil-

ity have been done using a real deployment of a cloud computing infrastructure. Although, [7] does an interesting approach Quality Models are not being investigated in the extension of them and also there is the lack of the statement of the possible quality dependencies between the functional layers discussed, PaaS and IaaS.

A platform-independent solution, which is called iLAND, has been proposed in [8] in order to support reconfiguration in service-oriented distributed soft real-time systems, in favor of time-bounded operations. iLAND middleware deals with specific use cases such as (a) the support of service oriented applications; (b) the integration of time-deterministic reconfiguration techniques and service-composition algorithms; (c) real-time communications by defining the complete network protocol stack (i.e., time-triggered level-2 media access control to enable schedulability analysis); and (d) portability to different communication off-the-shelf middleware. Architecture of iLAND is mainly based on three functional layers:

- The Core Functionality Layer (CFL) contains most of the key added-value functionality related to application management and reconfiguration control. It is supported by a number of managers responsible for the reconfiguration
- The Communication Backbone and Resource Management Layer (CBL) where the managers of it are responsible for the re-distribution of the communication data and
- The Network Layer (NL) contains the basic functionality for real-time transmission on general networks that offer TCP/UDP over IP. An application-specific communication protocol can be used in the specific module, such as streaming communication with specialized protocols as RTP enhanced with RTCP-based communication at transport level.

As the main focus of [8] is on real-time monitoring and reconfiguration, the fact that the services might be a functional piece of a workflow has not been taken into account. Researchers are mostly based on service-oriented applications, which could help in the possible re-organization of the services themselves in order to function properly.

State-of-the-art research provides some approaches towards cross-layer monitoring. Kazhamiakin et al. [9] define appropriate mechanisms and techniques to address monitoring in an integrated cross-layer framework. More precisely, they have stated the problem of cross-layer SBA monitoring and adaptation on a series of case studies and define the requirements for the integrated approaches that provide coherent solutions to monitor and adapt the whole application. Although, [9] suggests a monitoring solution that deals with the three layers of Business, Service and Infrastructure there is the lack of the definition of the Workflow QM which is , as also stated before , a major functional layer that has to be analyzed prior to the devolution on the Business layer.

In [10], authors present an integrated approach for monitoring and adapting multi-layered SBAs. The approach coprises four main steps: 1) monitoring and correlation, 2) analysis of adaptation needs, 3) identification of multi-layer adaptation strategies and 4) adaptation enactment. In the Monitoring and Correlation step, the sensors that are being deployed capture run-time data about its software and infrastructural elements in order

to be aggregated and correlated under the form of general and domain-specific metrics. Their main goal is to reveal correlations between what is being observed at the software and at the infrastructure layer to enable global system reasoning. This approach is mostly based on a run time dependency (monitoring) data mining between the layers of service and infrastructure. In our work, in addition to the dependencies that are being modeled in the according Dependency QM and captured at run-time, we have also stated Workflow and Service dependencies between the quality metrics that are being defined in each of according QMs, stating this way a holistic view of the relationships between the different functional layers. In order to tackle also the need for adaptation, in our previous work [1] we propose a monitoring framework for Multi-Cloud SBAs, that is able to perform cross-layer (Cloud and SOA) monitoring enabling concerted and majorally based upon adaptation actions.

Many layer-specific approaches regarding QMs, [11] [12] are based on stochastic models and probabilistic theory having as a major concern the scalability of the Cloud resources based on quality metric results. Several approaches have been proposed capturing infrastructure QMs with focus on Cloud resources. Authors in [13] define QMs which support the evaluation of public Cloud services; the validation of these QMs is performed according to empirical case studies without taking into account the relations that WM, IM and SM can have between them. Precise definitions of scalability and elasticity are given in [14] and [15], spanning mostly the service and infrastructure layers; our work on the other hand defines scalability and elasticity metric dependencies between those two layers and does not cope with them in isolation.

Finally, in [16] a layer-specific extensive WM is proposed for the specification of workflow QoS, as well as methods to analyze and monitor QoS. Specifically, two points are being issued:

- A presentation of a comprehensive model for the specification of workflow QoS as well as methods to compute and predict QoS is based on (a) the investigation of the relevant QoS dimensions that are necessary to correctly characterize workflows (b) on the various quality dimensions that are required in order to develop a usable workflow QoS model and (c) on algorithms that are being used in order to compute the QoS of workflows. The computation of workflow QoS is done based on the QoS of the tasks that compose a workflow. In our approach we have also been based on the tasks action in order to compute metrics that deal with workflow instances, but whenever necessary we completely separated the notion of workflow and task in order to compute the according quality metrics, for example the computation separation of task fidelity and workflow fidelity is something that gave a clear value to our monitoring data results.

- The second issue that has been stated by the researchers of [16] describe the enhancements that need to be made to workflow systems to support processes constrained by QoS requirements. These enhancements deal with (a) the implementation of a QoS model, (b) the implementation of algorithms to compute and predict workflow QoS, (c) the implementation of methods to record and manage QoS metrics. The workflow management system [17], that is used in order to a adapt these

enhancements is based on a technology that is not used widely anymore by the research community, whilst in our approach we have been based on an Worklfow Processs management [18] system that is being widely accepted by the industrial and research community and deals with structured BPMN processes.

In [19], the authors introduce the hypothesis that reliability of workflows can be notably improved by advocating scientists to preserve a minimal set of information that is essential to assist the interpretations of these workflows and hence improve their reproducibility and reusability. More precisely two quality dimensions were approached, (a) the stability of a workflow defined as a measurement of the ability of a workflow to preserve its properties through time and (b) the reliability of a workflow defined as a measurement for converging towards a scenario free of decay, i.e. complete and stable through time. Therefore, researchers combine both measures completeness and stability in order to provide insight into the behavior of the workflow and its expected reliability in the future. In our case, beside the clear separation of workflow and task reliability ,which drives us to a solid separation of workflow and task quality metrics, we also take into account user requirements based on the level of the fidelity a.k.a reliability of task or workflow. Even more in our approach we also consider quality dimensions of time, cost and security providing this way an extended QM able to describe a number of quality aspects by the definition of the appropriate metrics.

In [20] an infrastructure for monitoring applications deployed in virtualized environments, is presented. The infrastructure introduces two monitoring layers on top of the given standard monitoring models in order to provide enhanced security and dependability for cloud computing. In this research two QMs were defined; (a) a SaaS QM compatible with ISO/IEC 25010 standard [21] and (b) a Run Time quality model which specified the monitoring requirements, metrics, operationalizations, and configurations that were used during the monitoring procedure. From the other hand, our approach is an integrated solution and predominates the one of [20] for two reasons , 1) in our QMs we cope with computation formulas and definition of metrics for each of the three functional layers and 2) we approach the notion of Workflow Monitoring, issues that were not taken into account by [20], which are drawbacks considering the fact that the main customer of Virtualized infrastructure environments are SMEs.

2.1.2 Comparison

In order to state the advantages and disadvantages of the research approaches, we have made a comparison among them, based on certain criteria. These criteria indicate the subsistence of quality metrics for each of the functional layers, the existence of cross-layer dependency and the provision of monitoring solution that represent the according research approach. We can observe from Table 2.1 that almost all of the research approaches deliver a monitoring solution and only three of them deliver, partially, a dependency model; for example [8, Marisol et al. 2013] designate the dependencies for non cross-layer metrics but within the model of SaaS and [10, Guinea et al. 2011] states the run-time dependencies between Service and Infrastructure layers, without being based

Criteria / Research Approach	**Business Layer**	**Workflow Layer**	**Infrastructure Layer**	**Service Layer**	**Cross-Layer Metric Dependencies**	**Monitoring Solution Offered**
[9, Kazhamiakin et al. 2009]	✓		✓	✓		✓
[5, Bratanis et al. 2010]	✓		✓	✓		
[10, Guinea et al. 2011]			✓	✓	*partially*	✓
[3, Bratanis et al. 2011]	✓		✓	✓	*partially*	✓
[19, Gomez et al. 2013]		✓				✓
[8, Marisol et al. 2013]			✓	✓	*partially*	✓
[6, Alhamazani et al. 2015]			✓	✓		✓
[7, Calero et al. 2015]			✓	✓		✓
[20, Cedilo et al. 2015]			✓	✓		✓
[Metallidis et al. 2016]		✓	✓	✓	✓	✓

Table 2.1: Comparison of Research Approaches based on certain criteria

on solid quality dependency model between the layers. From the other hand, only our approach,[Metallidis et al. 2016], provide a solid representation of the three functional layers by the definition of a QM for each one of them, a dependency quality model that states the dependency relationships between the quality metrics of the different layers and a distributed monitoring solution that supports each of the QMs. We should also mention that the Business layer, is not part of our contribution but research approaches like [9, Kazhamiakin et al. 2009], [5, Bratanis et al. 2010] and [3, Bratanis et al. 2011] took it into account by defining and using quality terms that have to do with the Business layers, like the notion of a KPIs [22]. As we are stating in the Chapter 8 of Future Work, we intend to extend our monitoring solution and develop a QM in order to shore up the Business layer, offering these way a completely distributed monitoring solution that will cover all of the layers.

Chapter 3

Quality Models

Quality dimensions cover an important aspect of quality models involving dimension-specific attributes that can be measured by one or more respective dimension metrics. Calculation formulas, quality metrics and particular types of metric relationships are being defined in a formal way in order to indicate the structure of the according layer. We integrate quality models by the means of mapping quality metrics at different layers and by having as initial guide the quality dimensions so as to find candidate metrics at a lower layer that could be connected/mapped to the metrics at the (next) higher level. Catering for the workflow based usage of a service as well as the infrastructure that supports software components realising part of the workflow functionality, we take in account three main quality models: (a) the *Workflow Quality Model* stressing quality terms related to tasks and workflows, (b) the *Service Quality Model* indicating quality terms for assessing the quality of services, and (c) the Infrastructure Quality Model encompassing quality terms suitable for the assessment of the quality of the underlying cloud infrastructure. Additionally, to produce suitable and complete quality models which can be indeed easily exploited by cross-layer monitoring systems, the quality models included in them should satisfy the properties of measurability, validity and definition formalization able to characterize the quality of the according layer. We have structurally separated the quality terms in four categories:

1. **Quality Dimensions**: Describes the quality aspect which can be used to provide an aspect-specific partition of quality terms. Quality dimensions are independent of the layer that a term maps to.

2. **Quality Attributes**: Quality attributes are properties of an object which are measured by metrics.

3. **Raw Quality Metrics**: They represent raw information that could be taken from sensors being deployed in any of the three different layers (we do not take into account the procedure that the sensors take in order to calculate the raw metrics as this procedure is mainly software dependent)

4. **Composite Quality Metrics**: Metrics, raw or composite, can be used in order to

calculate composite metrics by applying a specific metric formula that has to be defined. In particular, we have two types of aggregation/calculation:

(a) application of statistical functions over the measurements of a metric and

(b) application of mathematical operators over the current measurement of one ore more metrics

Following the above classification we have a clear view of how a quality metric's measurements are going to be calculated from computational formulas applied over same or lower level measurements. Before we continue with the definition of the QMs that we are going to use on the cross-layer monitoring system we should also mention that we are based on characteristics stated in [23] that each of the quality metrics should have, see Section 5.6.

3.1 Workflow Quality Model

We have defined a quality model for the **Workflow** layer which contains workflow and task metrics as advocated in [16]. It also explicates how the task measurements can be propagated to the level of workflow to produce the respective measurements of workflow metrics through metric aggregations. The proposed quality model comprises four quality dimensions of time, cost, reliability and security.

3.1.1 Time Quality Dimension

Quality dimension of **time** is a fundamental aspect of performance that describes the time needed in order to measure, execute, record, respond and traverse through operations.

For the Workflow level we define the following metrics:

- *Workflow Process Time (WPT)*: This parent metric depicts the ultimate (externally observed) time needed by a workflow to be processed. It includes the execution and delay time over all tasks along with the delay time needed to transition from one task to another indicating the fact that WPT equals to the addition of WET, WDT and WTDT metrics described below.
  - *Workflow Execution Time (WET)*: This metric calculates the externally observed execution time of the workflow which is based on the structure and the execution time of its components (tasks and sub-workflows). We consider only the pure task execution time and not any type of delay in order to calculate WET.
  - *Workflow Delay Time (WDT)*: Metric of WDT depicts the resulting externally observed delay time that depends on the workflow structure and is calculated based on each of workflow's task's pure/intrinsic delay time (excluding its execution and (onwards or backwards) transition time).

  - *Workflow Transition Delay Time (WTDT)*: WTDT metric indicates the aggregated externally observed transition delay time of the workflow calculated from the time each workflow transition takes to move from one task to another.

- *Overall Workflow Execution Time (OWET)*: This metric indicates the overall execution time being spent when executing all tasks in the workflow which is independent from the workflow structure.

Workflow metrics can be calculated based on formulas over task metrics and (transition) metrics mapping to pair of tasks. Some of them can be viewed either as raw in case the workflow monitoring system provides sensors to directly measure them or composite if they are produced from other metrics. One way or the other could be preferred depending on the available sensors exploited by the monitoring system; for example the calculation of WPT could rely on the addition of WET, WDT and WTDT or can be based on one sensor included within the workflow execution engine able to sense and subtract the end and start time points of the respective workflow instance. In case of task metrics the same choices are applied;some can be measured via sensors and other via lower-layer metrics (e.g., task execution time from response time of the external service executed). We also want to stress that the workflow structure has an impact on the calculation of certain workflow metrics. For instance, for a parallel structure, the workflow execution time would be the max execution time across all parallel execution paths/branches of this structure. We are defining the following component metrics in the task level:

- *Transition-Delay Time (TransDT)*: Defines the delay time that corresponds to the transition from one task to another task, where the transition can be direct or indirect. A direct transition is the immediate transition from one task to another and the indirect one is the one where two or more delays are involved. In the indirect calculation, we consider the transition of pair of tasks; for example based on the Figure 3.1 for the transition of Task_3 to Task_6 we calculate $\text{Task}_3 \rightarrow$ Gateway, Gateway $\rightarrow \text{Task}_6$ such that we actually consider $\text{Task}_3 \rightarrow \text{Task}_6$. We apply this computation considering all the gateways placed as mediators in the process of the indirect calculation of the transition delays. Thus, the metric of TransDT can be calculated by placing a sensor that senses the finishing time of the previous task and the start time of the next one and then computes the difference in order to compute the transition delay time for this pair of tasks. A pair of tasks is directly or indirectly identified by the execution flow if in the workflow specification, Task_c immediately follows Task_b (where immediately maps to not having any task in between but allows to have gateways as mediators); then the transition between these two tasks will be taken into account mapping to a valid task pair, where b and c are defined as Task numbers.

- *Task Delay Time (TDT)*: Defines the amount of time spent between the task enactment and its actual execution starting point. Additionally, it also covers the time between the execution end and the task end. Thus, this metric can be split into:

  - *Management Delay Time (MDT)*: This raw metric refers to the preperation and finishing time needed for a task due to respective suitable actions performed by the workflow management system.
  - *Process Delay Time (PDT)*: This composite metric refers to the time delay from the moment the task is waiting on queue of tasks to the moment that its execution starts. To measure this delay we can define two additional component metrics.
    * *Queuing Delay time (QDT)*: This metric defines the time the task has spent waiting in a tasklist, until it is assigned to an entity.
    * *start Entity Delay Time (sEDT)*: This metric refers to the time after the assignment of a task to the respective entity until the actual execution starts.
    * *end Entity Delay Time (eEDT)*: Similar to sEDT, this metric refers to the time after the actual execution of the task until the time it responds to the workflow management system.

- *Task Execution Time (TExT)*: Defines the amount of time spent to perform the task by any entity, e.g., a human-based or software component. To compute this execution time, we need to have indications when the task execution started and ended. In case of human-based tasks, it is the respective human that informs the system about this information. In case of service/automatic tasks the execution time can be accurately measured by the Workflow Engine as it takes care of executing respective internal or external software/service mapping to these tasks. Thus, in order to allow for the complete derivation of the TExT metric in all possible cases, a connection between workflow and service quality model has to be established.

3.1.1.1 Workflow Timing metrics calculation

We need to stress here that for the computation of the workflow metrics we consider the structure of the workflow instance being run by mapping it into a reduced workflow model representing a respective execution graph, In this sense, we can neglect particular workflow constructs, like conditional ones, by reducing them to the suitable selected substructure, as we possess the runtime knowledge of the actual flow of execution. The following algorithm handles the calculation of the WET by taking into account the remaining sequential and parallel structures within the workflow instance's runtime execution path. In order to calculate OWET metric we just add the execution time of all the tasks following the formulation of , $\text{TExT}_1 + \ldots + \text{TExT}_N$, where N is the number of tasks of the according workflow instance.

To assist in the calculation of WET we have defined a composite metric called sub-WorkflowExecution (sub-WE) which represents the execution time of a workflow's substructure/sub-element. Possible values of the sub-WE metric depend on the type of the respective sub-structure. A sub-structure could be any sequential or parallel structure within a workflow. In the case of a sequential structure, the respective sub-WE metric is computed from the addition of TExT and sub-WE values mapping to the contents of

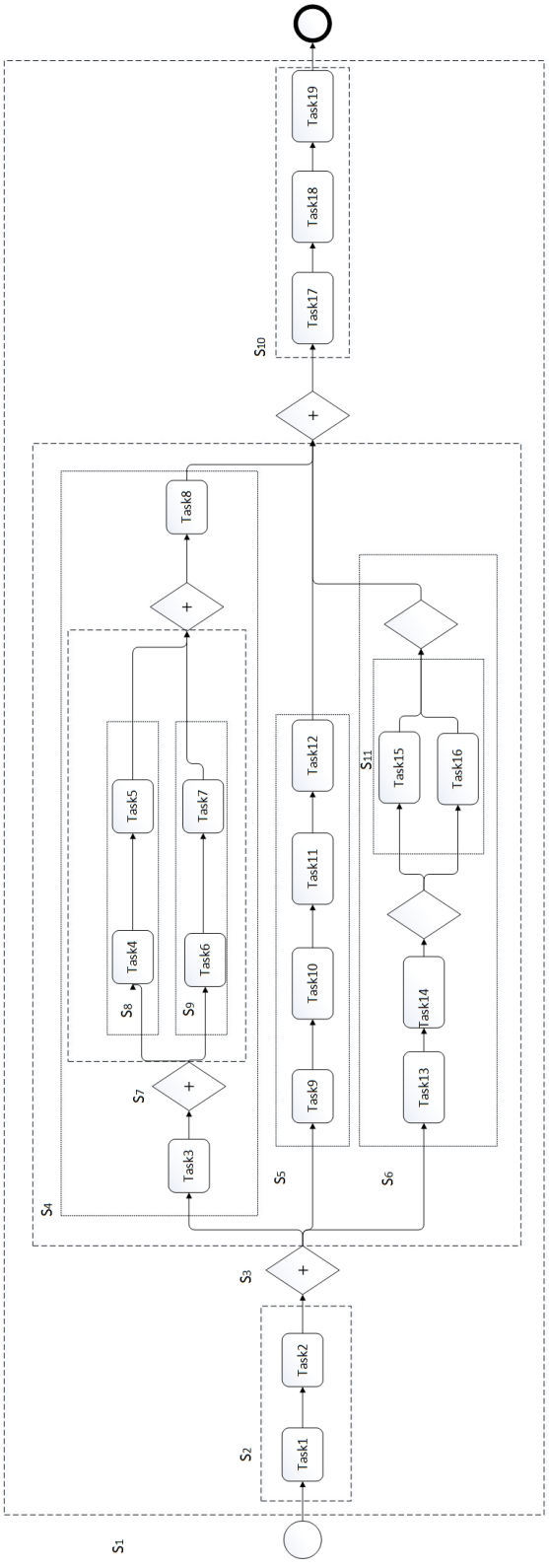


Figure 3.1: BPMN Workflow indicating sub-WE elements

Metrics for the Calculation of the Workflow Execution Time	Calculation Formulas
Workflow Execution Time(WET)	sub-WE$_1$
sub-WE$_i$	Sequential Element Case: TExT$_{i,1}$ + ... + TExT$_{K,N}$ + sub-WE$_{1,2}$ + ...+ sub-WE$_{K-1,K}$ Parallel Element Case: max(sub-WE$_{1,2}$,...,sub-WE$_{K-1,K}$, TExT$_{i,1}$,...,TExT$_{K,N}$)

Table 3.1: Formulas for the calculation of WET.

this structure (i.e., tasks and internal sub-structures), respectively. In the case of a parallel structure, the sub-WE metric depends recursively on the max execution time value of the path branches included in this structure. The value of a sub-WE is computed once all the values/measurements for the components of it, are available. A new sub-WE is defined in three main situations:

1. For the single global/outer structure of the workflow which is equal to the workoflow's WET.

2. When next step in the workflow model is a *parallel* structure.

3. When next step in the workflow is a *sequential* structure;

In Figure 3.1 we are providing a slightly complex BPMN workflow model which can reflect the reality and which is mainly used for explicating the computation procedure for workflow metrics. This workflow includes eight sequential structures, two parallel structures, and a conditional structure. *S* symbol represents any kind of structure in the BPMN diagram. In case of a conditional structure, we just follow the actual runtime structure to reduce the conditional elements to those elements that have been actually followed/selected. In this sense, Task$_{15}$ instead of Task$_{16}$ is followed/selected to be executed for S_1. Main purpose of the example workflow is to explicate how the proposed algorithm is applied to a specific workflow in order to compute values of the respective workflow metrics. We use it in order to depict the respective suitability of the metrics defined as well as unveil the way they are calculated. In this layer, metric values that can be calculated based on the proposed algorithm are the WET, WDT and WTDT metrics.

In Table 3.1, *i* indicates a number between 1 and *K*, where *K* is the number of sub-structures which maps to sub-WE metrics and *N* is the number of tasks which maps to the TExT metrics (as has been stated before). More precisely, we rely on a procedure which leads to the production of a simple computation formula. This formula is recursively broken down into additional components, so as to compute the WET metric of a specific workflow. In order to compute each of the sub-WE metrics we have the convention of sub-WE$_1$ represents the execution time spent in structure S$_1$, sub-WE$_{1,2}$ represents the

Workflow Delay Time metrics	Calculation Formulas
Workflow Delay Time(WET)	sub-WD_1
sub-WD_i	Sequential Element Case: $TDT_{i,1}$ + ... + $TDT_{K,N}$ + sub-$WD_{1,2}$ + ... + sub-$WD_{K-1,K}$ Parallel Element Case: max(sub-$WD_{1,2}$,...,sub-$WD_{K-1,K}$, $TDT_{i,1}$,...,$TDT_{K,N}$)

Table 3.2: Formulas for the calculation of Workflow Delay Time.

execution time spent in structure S_2 and so on. Each of the tasks/sub-elements are being indexed by two numbers *i*, *j*, where i is the number of the sub-structure containing task/sub-structure with number j. In case of the execution time of tasks, $TExT_{2,1}$ represents the execution time spent in $Task_1$, $TExT_{1,2}$ represents the execution time spent in $Task_2$ and so on. Thus, following the above rules and Table 3.1 formulations, we have the following assignments:

- WET = sub-WE_1
- sub-WE_1 = sub-$WE_{1,2}$+ sub-$WE_{1,3}$ +sub-$WE_{1,10}$
- sub-$WE_{1,2}$ = $TExT_{2,1}$ + $TExT_{2,2}$
- sub-$WE_{1,3}$= max(sub-$WE_{3,4}$,sub-$WE_{3,5}$, sub-$WE_{3,6}$)
- sub-$WE_{3,6}$ = $TExT_{6,13}$ + $TExT_{6,14}$ + $TExT_{6,15}$
- sub-$WE_{3,5}$ = $TExT_{5,9}$ + $TExT_{5,10}$ + $TExT_{5,11}$ + $TExT_{5,12}$
- sub-$WE_{3,4}$ = $TExT_{4,3}$ + sub-$WE_{4,7}$ + $TExT_{4,8}$
- sub-$WE_{4,7}$ = max (sub-$WE_{7,8}$,sub-$WE_{7,9}$)
- sub-$WE_{7,8}$ = $TExT_{8,4}$ + $TExT_{8,5}$
- sub-$WE_{7,9}$ = $TExT_{9,6}$ + $TExT_{9,7}$
- sub-$WE_{1,10}$ = $TExT_{10,17}$ + $TExT_{10,18}$ + $TExT_{10,19}$

The same procedure is going to be followed for the calculation of the WDT where we are going to have TDT values for each of the tasks and sub-WD instead of sub-WE for the workflow (sub-)structures. In order to calculate Workflow Delay Time we have defined the metric of sub-workflow Delay, sub-WD. To calculate WDT and sub-WD we have specified the equations shown in Table 3.2.

Regarding the calculation of TransDT and WTDT we have followed a similar calculation procedure as the one previously highlighted. This time we define sub-WorkflowTransitionDelay

Workflow Transistion Delay Time metrics	Calculation Formulas
Worklfow Transition Delay Time(WTDT)	sub-WTD$_1$
sub-WTD$_i$	Sequential Element Case: TransDT$_{i,1}$ + ... + TransDT$_{i,n+1+2*k}$ + sub-WTD$_{1,2}$ + ...+ sub-WTD$_{K-1,K}$ Parallel Element Case: max(sub-WTD$_{i,1}$,...,sub-WTD$_{K-1,K}$, TransDT$_{i,1}$,...,TransDT$_{n+1+2*k}$)

Table 3.3: Formulas for the calculation of Workflow Transition Delay Time.

(sub-WTD) in order to group transition delays in the same way as done for Execution and Delay Time. The main difference in calculation regards the fact that different components are now considered. Computation is going to take place in two situations:

1. In case the sub-WTD metric maps to a sequential structure then it would be equal to the addition of the TransDT and sub-WDT metrics. Internally, each sub-structure have tasks and/or sub-elements mapping to a sub-WTD metric. Thus, in a sequential structure, the number of transitions is n+1+2*k. Where n is the aggregated number of tasks from each of the sequential sub-structure that does not contain a parallel structure and k is the number of sub-structures that are being constituted by parallel structures. The reason that we multiply the number of the parallel sub-structures by a number of two transitions is that we define a sub-WDT as a transition delay too, thus it add two more transitions delays in the computation procedure.

2. In case the sub-WTD metric reflects to a parallel structure then we consider the max value among all branches/flows which reflects on one of the two cases indicated:

   (a) if a parallel branch maps to one task, then we have two transitions (to reach and leave the task);

   (b) if a parallel branch maps to a sub-structure then we have two transitions plus sub-WTD (to reach and leave the sub-structure).

Table 3.3 indicates the procedure that leads to the production of a recursive computation formula for the WTDT metric of a specific workflow; sub-WTD_1 represents the delay time of structure S_1, sub-WTD$_2$ represents the delay time of structure S_2 and so on. In case of the transition delay between two workflow elements, TransDTD_1 represents D_1, TransDTD_2 represents D_2 and so on. Thus, following the above rules and table 3.3 we are going to have the following assignments to calculate the WTDT for Figure 3.2 :

- WTDT = sub-WTD$_1$
- sub-WTD$_1$ = sub-WTD$_{1,2}$+ sub-WTD$_{1,3}$ +sub-WTD$_{1,10}$

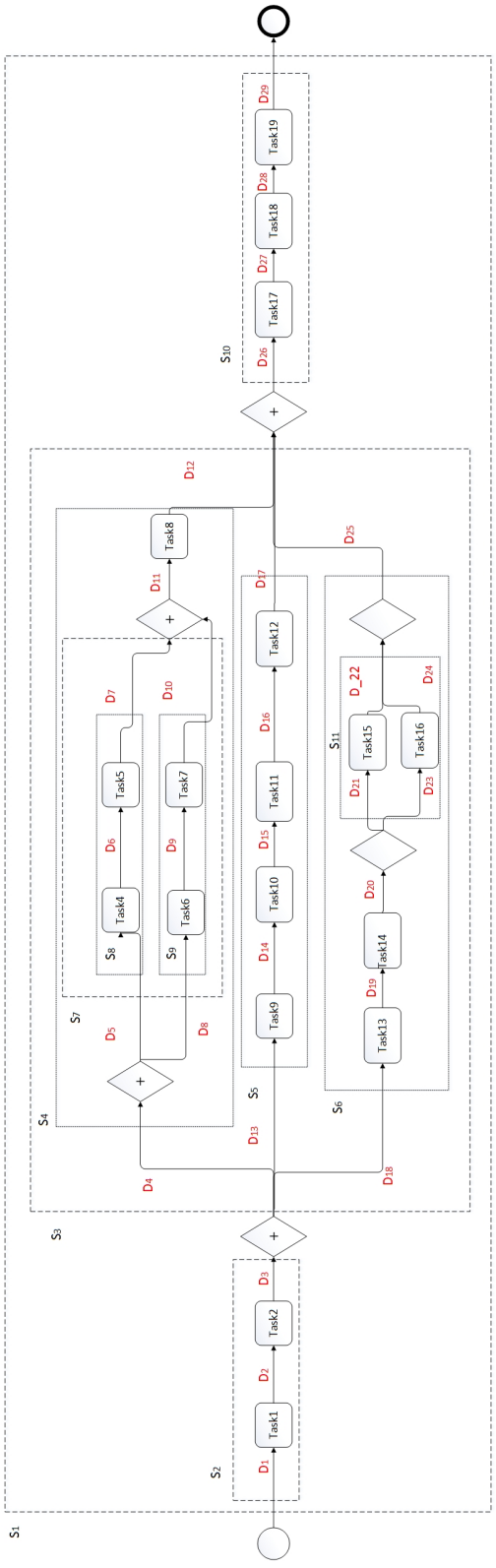


Figure 3.2: Workflow Transition Delay Diagram

- sub-WTD$_{1,2}$ = TransDT$D_{2,1}$ + TransDT$D_{2,2}$ + TransDT$D_{2,3}$
- sub-WTD$_{1,3}$ = max(sub-WTD$_{3,4}$,sub-WTD$_{3,5}$,sub-WTD$_{3,6}$)
- sub-WTD$_{3,6}$ = TransDT$D_{6,18}$ + TransDT$D_{6,19}$ + TransDT$D_{6,20}$ + TransDT$D_{6,21}$ + TransDT$D_{6,22}$+ TransDT$D_{6,25}$
- sub-WTD$_{3,5}$ = TransDT$D_{5,13}$+TransDT$D_{5,14}$+TransDT$D_{5,15}$ + TransDT$D_{5,16}$+ TransDT$D_{5,17}$
- sub-WTD$_{3,4}$ = TransDT$D_{4,4}$ + sub-WDT$_{4,7}$ + TransDT$D_{4,11}$+ TransDT$D_{4,12}$
- sub-WTD$_{4,7}$ = max(sub-WTD$_{7,8}$,sub-WTD$_{7,9}$)
- sub-WTD$_{7,8}$ = TransDT$D_{8,5}$ + TransDT$D_{8,6}$ + TransDT$D_{8,7}$
- sub-WTD$_{7,9}$ = TransDT$D_{9,8}$ + TransDT$D_{9,9}$ + TransDT$D_{9,10}$
- sub-WTD$_{1,10}$ = TransDT$D_{10,26}$ + TransDT$D_{10,27}$ + TransDT$D_{10,28}$ + TransDT$D_{10,29}$

3.1.2 Cost Quality Dimension

Cost quality dimension represents the cost being associated with operations(e.g the provision of VM) and the execution of procedures. It is an important factor to evaluate whether organisation still follows precisely their financial plan.

We have defined the following workflow metrics:

- *Workflow Cost (WC)*: This metric depicts the cost of the entire workflow and is calculated by the addition of the costs of the actual tasks.
- *Workflow Engine Cost (WEC)*: WEC indicates the cost for running the workflow instance calculated by the scope of the workflow engine, with regard to the resources spent by the workflow engine. Metrics of it are:
  - *Managing Workflow Instance Cost (MWIC)*: Cost related with the effort of the Workflow Engine to manage the according Workflow Instance
  - *Resources Transfer Cost (RTC)*: This cost is related with the transfer of resources(e.g digital data) from one task to another.
  - *Resources Calculation Cost (RCC)*: Cost of RCC is related with the effort spent in order to calculate specific formulations of measurements.

To continue we have also defined a metric that depicts respective cost being related with a workflow task:

- *Task Cost (TC)*: Indicates the overall cost of a task. It can be calculated by the sum of TEC and TExC component metrics which are analysed below:

Workflow and Task metrics	Calculation Formulas
Workflow Engine Cost (WEC)	$MWIC_j$ + RTC_j + RCC_j
Workflow Cost (WC)	$Sum_{i=1}^{N}$ TC_i
Task Cost (TC_i)	TEC_i + $TExC_i$
Task Enactment Cost (TEC_i)	SuC_i +TdC_i
Task Execution Cost ($TExC_i$)	DLC_i+Mc_i + ARC_i

Table 3.4: Calculation of the composite Workflow Cost metrics.

- *Task Enactment Cost (TEC)*: Is the cost associated with the management of the workflow instances with respect to the corresponding workflow management system. This metric could be calculated by the sum of the following raw metrics
  - *Set up Cost (SuC)*: Cost to set up any resource used prior to the execution of a workflow task
  - *Tear down Cost (TdC)*: Cost to tear down any resource used after the execution of a workflow task
- *Task Execution Cost (TExC)*: Is the cost associated with the runtime execution of a task, within the scope of a particular workflow instance, that can be calculated by the following raw metrics
  - *Direct Labor Cost (DLC)*: This cost is associated with the person carrying out the execution of a workflow human task
  - *Machine Cost (MC)*: MC deals with internal and external resources associated with the execution of an automatic task. Internal resources within the organization running the workflow could be a locally computing center performing upon workflow task's and external are resources out of the respective organization running the workflow e.g additional computing power provisioning a Virtual Machine (VM) in a public cloud provider.
  - *Additional Resources Cost (ARC)*: We can have two types of additional resources : (a) humans which assist the one associated to a human task and (b) software which supports the execution of a human task.

The intuitive separation of the enactment and execution cost metrics is that the first one gives the cost of the task that is related with actions that have to do with the workflow management system which do not include any task execution, such as the set up cost of a resource; on the other hand, the second one is only coupled with the actual task execution related to the cost of the resources used to perform or assist in this execution, thus separating it strongly from the enactment cost which is a-priori or a-posteriori cost of the execution task procedure.

For the calculation of WC in Figure 3.3, have followed the equations shown in Table 3.4 (where $1 \leq i <$(N=19) and j is the number of the workflow instances of an according workflow).

3.1.3 Reliability Quality Dimension

Reliability dimension corresponds to the likelihood that a component (e.g workflow, task, service) will not malfunction or fail.

We have defined the following metrics:

- *Workflow Availability (WA)*: Metric of WA indicates the percentage of time that the workflow is available.
- *Workflow Fidelity (WF)*: Workflow Fidelity metric is used by the workflow system to compute how well workflow instances satisfy the user quality requirements within a specified period time of reference. Thus, we can measure Workflow Fidelity by applying the values of workflow metric measurements (mapping to the specified period of time) to a utility function which depends on users requirements.
- *Total Workflow Failure Time (TWFT)*: Metric of TWFT defines the total aggregated workflow failure time indicating the sum of workflow failure time (WFT) of instances of the same workflow that have finished (successfully or not). By defining TWFT we can also define the following:
  - *Workflow Failure Time (WFT)*: Workflow Failure Time metric is defined as the overall time where the tasks endured and suffered from failures for a certain instance of the workflow that has been stated as finished (successfully or not).
  - *Total Instance Workflow Failure Time (TIWFT)*: This metrics indicates the addition time of all the tasks happened to be in failure state, for a certain workflow instance that has finished it's operation (successfully or not). In the calculation of TIWFT we do not take in account the structure of the workflow elements

Definitions of the task metrics that have been defined are:

- *Task Failure Rate (TFR)*: Metric of Task Failure Rate indicates the rate that a certain task fails to operate properly for a certain workflow instance.
- *Task Fidelity(TF)*: Task Fidelity metric is used by the workflow system to compute how well tasks instances meet user requirements (at the task level) within a specified period time of reference.
- *Task Availability(TA)*: This quality metric indicates the percentage of time that a task is available. TA also depends on human availability upon a task whenever applicable. It can be measured by the respective systems which either senses or allows the human to declare his/her availability. Analyzing this quality metric can depend on information provided by the following three metrics:
  - *Mean Time To Recover (MTTR)*: Mean time to Recover from a failure and factor unavailability of a human. To calculate this we defined the following raw metrics:

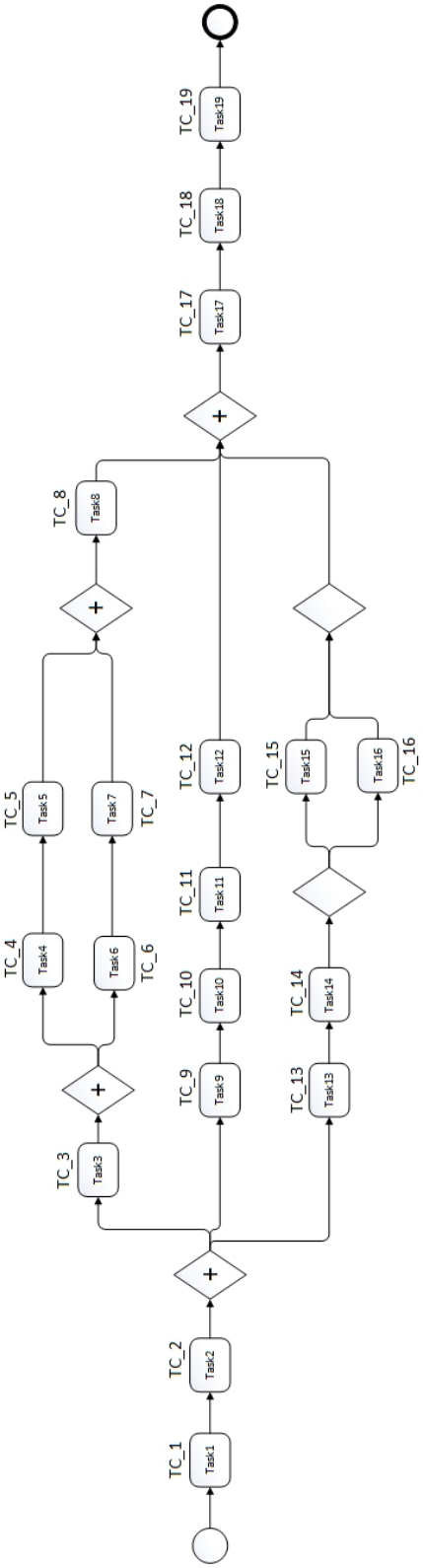


Figure 3.3: Workflow Cost Diagram

Workflow and Task metrics	**Calculation Formulas**
Total Workflow Failure Time (TWFT)	$\text{WFT}_1 + \ldots + \text{WFT}_N$
Workflow Failure Time (WFT)	Refer to Table 3.1. The algorithm computation is the same as in WET and WDT computations but in this procedure we calculate WFT and TFT instead of TExT and sub-WF instead of sub-WE, defined accordingly
Total Instance Workflow Failure Time (TIWFT)	$\text{TFT}_{i1} + \ldots + \text{TFT}_{iN}$
Workflow Availability (WA)	$\text{TA}_{i1} * \ldots * \text{TA}_{iN}$
Workflow Fidelity(WF)	$f_{WF}(hist, reqs)$
$f_{WF}(hist, reqs)$	$\frac{(sat(meas_1,reqs)+\ldots+sat(meas_A,reqs))}{A}$
$sat\,(meas_i, reqs)$	if $reqs.lowThreshold \leq meas_i.value \wedge meas_i.value \leq reqs.upperThreshold$ return 1; else return 0;
Task Availability(TA)	$\frac{MTBF}{MTBF+MTTR}$
Mean Time Between Failures(MTBF)	$\frac{TotalUpTime}{NumberofBreakdowns}$
Mean Time to Recover(MTTR)	$\frac{DownTime+HumanUnavailabilityTime}{NumberofBreakdowns}$
Task Failure Rate(TFR)	$\frac{1}{MTBF}$
Task Fidelity(TF)	$f_{TF}(hist, reqs)$
$f_{TF}(hist, reqs)$	$\frac{(sat(meas_1,reqs)+\ldots+sat(meas_B,reqs))}{B}$

Table 3.5: Calculation formulas of Workflow and Task Reliability metrics

  - *Down Time*: Is the raw metric of the down time for a certain period of time
  - *Human Unavailability Time*: Is the raw human down time metric for a certain period of time, differentiates from Down Time

- *Mean Time Between Failures (MTBF)*: Mean time between failures of a task. Raw metric needed for the calculation of MTBF is the following:
  - *Total Up Time*: Is the raw metric of the total up time for a certain period of time
- *Number of Breakdowns*: Is the number of breakdowns that the task encountered for a certain period of time.

In Table 3.5, we define the Workflow Fidelity formula. This formula maps to the function of $f_{WF}(hist, reqs)$. Symbol A equals with the number of workflow measurements, B with the number of task measurements and N with the number of tasks. More precisely we have the following definitions for the functions shown in Table 3.5:

- $f_{WF}(hist, reqs)$: Function WF_f takes as input two parameters. The first parameter depicts a set of metrics along with their measurement values mapping to the history time range of interest; while the second parameter represents the *user constrains*. A measurement is composed of metric,value and timestamp whilst the *user constraints* represent a threshold being applied over a specific metric. This function returns a value in the range [0.0,1.0] indicating the fidelity of the workflow.

- $f_{TF}(hist, reqs)$: This function initially selects those user constraints which map to the respective metric of the measurement and then performs the comparison of the measurement value against the user constraints' low and upper bound/thresholds. If the metric measurement is within these threshold, the output is 1; otherwise, it is equal to 0.

In the case of tasks, we independently calculate their fidelity in the context of a particular workflow by considering a specific period of time. We also want to stress that, we deal with two levels of constraints: (a) we have global QoS constraints on the workflow level and (b) local QoS constraints on the task level. The common case is that the global QoS constraints can involve many QoS metrics and that at the workflow level we have some QoS constraints on some QoS metrics for some tasks. In this sense the propagation of task's reliability with respect to the QoS constraints posed over the workflow reliability on global QoS constraints wouldn't be valid, thus we ended up calculating task and workflow reliability separately. Respective metrics and formulations are indicated in Table 3.5.

3.1.4 Security Quality Dimension

Quality dimension of **security** indicates the degree at which components and systems are free from unauthorized accesses or change, subject to policy. Security properties of metrics include auditability, authorisation, authenticity, confidentiality and non-reputability.

Below we define two metrics that characterizes the level of security for a workflow. The first metrics defines a global security metric that consumes information provided by the appropriate security system about the incidents that were spotted. The second metric defines a metric that calculates the action of proper and improper authentication, that is also being used as a security incident whenever the authentication is being breached.

- *Workflow Incident Rate(WIR)*: This metric defines the level at which a workflow has suffered from security incidents.

- *Workflow Authentication Security (WAuS)*: Metric defining the level at which the workflow has suffered from improper user authentications.

To compute the metric of WIR we have defined the following quality attribute for the task level:

- *Task Incident Rate (TIR)*: This metrics indicates the number of security incidents a task has experienced during a certain time window.

Workflow and Task metrics	Calculation Formulas
Workflow Incident Rate(WIR)	$\frac{TIR(t1)+...+TIR(tN)}{max(periodTine)}$
Workflow Authentication Secutiry (WAuS)	min(TAuS$_{(t1)}$, ..., TAuS$_{(tN)}$)
Task Incident Rate (TIR)	$\frac{NumberOfIncidents}{periodTime}$
Task Authentication Security (TAuS)	100 - $\frac{TFAu+TVAu}{TTAAuth}$ *100

Table 3.6: Calculation formulas of Workflow and Task Security metrics

  - Calculation of this metric equals with the amount of incidents being detected by the according security system divided by the time window which could be expressed as an absolute unit of time (seconds, hours or days).

Metric of WAuS could be calculated by the following metrics:

- *Task Authentication Security (TAuS)*: Metric of TAuS indicates the level at which the task is free from improper user authentications. Definitions of metrics are given below:
  - *Task Vulnerable Authentications (TVAu)*: Number of successful improper user authentications that the where reported by the according security system.
  - *Task Failed Authentications (TFAu)*: Metric of TFAu indicates the number of failed improper user authentications that were successfully addressed by the security system.
  - *Total Task Attempts for Authentication (TTAAuth)*: This metrics depicts the total attempts for authentication.

Calculation of the above metrics are shown in Table 3.6. We also want to stress that if the denominators of a certain formula task equals to zero then the task is considered to be secured from vulnerabilities.

3.2 Service Quality Model

Service Quality Model is based on metrics that depict the quality of the respective service. To be more precise, quality of a service maps to the quality that a user perceives when using the according services based on SLOs but also to the internal quality of a service (could be involved in event patterns guiding adaptive service behaviour) which maps to the capturing of the service provider view. We rely on *Performance* dimension metrics, indicating the performance of a web service, but also on *Stability* metrics that are related to the reliability, availability and accessibility of the service itself. Third and forth quality dimensions refers to *Scalability* and *Elasticity* metrics accordingly, where in this sense, we consider metrics which assess the ability of a service to scale in an autonomic manner based on the load received, as long as the degree of the scaling. Fifth dimension refers to

Security metrics that Service Quality model provides. To continue, we analyze these five quality dimensions with respect to the included metrics at this layer.

3.2.1 Performance Quality Dimension

The **Performance** dimension refers to the velocity of a Web service responding to any service request. Thus, we can define the following quality attributes and metrics:

- *Response Time*: Quality attribute of Response Time indicates the time taken to send a request and to receive the respective response. We are based on the navigation timing API [24] in order to compute network delays and page loading procedures. The following metrics have been defined:
  - *Request Completion Time(s)*: Is the time taken from the time that the response starts to load till the load event is completed. We calculate it based following equation [24] : *loadEventEnd - domLoading*
  - *Raw Response Time (s)*: For a single request we can define the metric of Response time which can be calculated by the sum of the following two metrics :
    - *Execution Time (s)*: Is the time taken for the software component to execute a single request:
      - *Process Time (s)*: Time take in order for the software component to process the respective metric request.
      - *Delay Time (s)*: This metrics indicates the delay time of the software component in order to start processing the metric request.
      - *Answer Delay Time (s)*: This metrics indicates the delay time of the software component in order to respond subsequently of the processing end.
    - *Network Latency (s)*: Indicates the time taken from the time the user starts searching relevant application caches till the last byte of the response is received. We exclude the execution time as we care only for the network latency traffic. We calculate network latency based on the following formulation [24] : *(connectEnd - fetchStart) + (responseEnd - responseStart)*
  - *Maximum Response Time (s)*: This metric indicates the maximum response time exhibited by a service within a specific time period of reference
- *Throughput*: Quality attribute of Throughput indicates the number of service requests that were processed. Metrics that characterize throughput are:
  - *Average Requests Number (p/m)*: Indicates the average number of requests for a service component within a specific unit of time

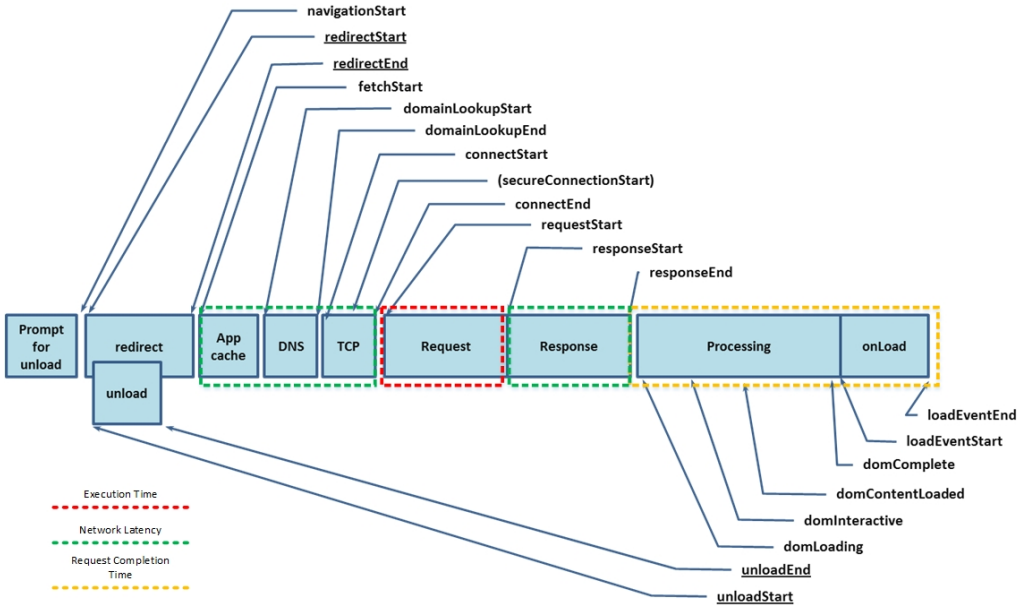


Figure 3.4: Navigation API

– *Maximum Throughput*: Maximum Throughput of a service indicates the maximum number of service requests that are being processed within a period of time

In Figure 3.4 we show which parts of the Navigation API [1] have been used on order to compute the aforementioned response time metrics.

3.2.2 Stability Quality Dimension

Stability quality dimension indicates the ability to provide reliable, continuous, adaptable, consistent and recoverable services despite undesired situations like increased load, congestion, system failure and natural disasters. This quality dimension has the following quality attributes:

- *Availability*: We can define the availability as the ratio of time in which a service is expected to function properly. We have to stress that we do not consider networking issues as these are related to service accessibility. In case of external services the availability can be measured based on reports that clients or third-part entities produce by assessing the availability of these services based on their uptime status (as conceived by these entities which could also be related to network issues that cannot be easily detected). Thus we defined the following metrics that refer to the availability of the service for a specified period of time:

[1] `https://www.w3.org/TR/navigation-timing/`

  - *Down Time (%)*: Depicts the average ratio of time where the the service component is down
  - *Availability (%)*: Is the percentage of time the service is available-use of a service depends also on its accessibility (which can be also related to whether it has reached its processing limits or not), which equals to *1 - Down Time*

- *Reliability*: The reliability quality attribute assesses how reliable is the service which is translated to having the least possible number of failures and the largest possible amount of time between them. Metrics of this quality attribute could be:

  - *Mean Time To Failure (hours/service) (MTTF)*: This metric depicts the mean time to failure in regard to the number of services that are being monitored. It is computed by the formula of $\frac{totalFailureTime}{NumberOfServicesMonitored}$.
  - *MTBF (hours/failure)*: Mean time between failures of a service is the average elapsed time between failures of a service during its operation. It can be computed by the calculation formula of $\frac{totalUpTime}{NumberOfFailures}$ of a specific service. The difference between MTBF and MTTF is that MTBF concerns a specific service whilst MTTF is a general metric indicating the time to failure of all the monitored services.
  - *Fidelity of a service*: Defines the degree with which a service satisfies the user quality requirements within a specified point time of reference
  - *Successability (%)*: is defined as the extent to which Web services yield successful results responses over valid request messages; we define as valid all requests that conform to the input message schema but can map to values that might not be acceptable by the according service (such that an error is reported). This metrics can be calculated by the following formula, $\frac{NResM}{NReqM}$; if the number of response messages is equal to that of request messages, then we consider that the service successability equals to one. The component metrics mapping to the successibility metric formula are:
    * *Number of Response Messages (NResM)*: Number of response messages delivered from the according software component.
    * *Number of Request Messages (NReqM)*: Number of messages that where requested to the service component in order to retrieve the requested information

- *Accessibility*: is defined as the degree that a system could properly being accessed. Accessibility relies on factors that has to do with network issues and the reaching of the processing limit of the service making it unable to accept additional requests. The respective metrics from which accessibility can be measured is:

  - *Accessibility value*: This value indicates the level at which the service component is accessible with respect to a reference time period. It is calculated by the division of the acknowledge messages with the total tries to access it. The following metrics are being defined in order to calculate the rate of access:

    - *NReqM*: Same definition as the one of the reliability attribute
    - *Number of Acknowledgements received*: Number of acknowledgements received by the software component. It's difference with the metric of NResM is that this metric takes into account only messages that where properly acknowledged (e.g status 200) whilst in NResM we do not take into account the type of the response as long as they exists.

- *Adaptability*: quality attribute of adaptability describes the capability of the service or the instantiated virtualized infrastructure component, to adapt on situations in order to follow the underlying system's and infrastructure specifications. We have define the following metrics in order to describe the adaptability:

    - *Adaptation Time (s)*: Time needed by the according software component in order to adapt on the underlying system, including possible infrastructure scaling actions e.g time taken in order the service component to be adapted on a new operating system.
    - *Precision of Adaptation*: It describes the degree at which the adaptation actions had as a result the alignment of the service or the virtualized infrastructure component with the new specifications of the according system that are being adapted to, based on certain SLOs. It is calculated by the number of times that the according software or virtualized infrastructure component had a precise aligned adaptation divided by the total times an adaptation for the according component.
    - *Successability of Adaptation(%)*: Describes the percentage of times that the action of the adaption was successful. We differentiate the successful actions of adaptation with the precision by the means that an adaptation is considered as successful when it is functional upon the new/upgraded/downgraded system that the service is based.

3.2.3 Scalability Quality Dimension

By relying on [15], Service **Scalability** is defined as the ability of a service to scale when additional workload is received to still keep up with the SLOs promised (e.g., when reaching maximum capacity for the service, the capability of the application encompassing additional resources in order to still satisfy its performance goals/SLAs would be essential). Metrics for the Service Scalability are:

- *Scaling Utilization*: Scaling Utilization refer to the ability of the underlying system to increase or decrease it's software resources with the support of the existing IaaS resources. It defines the factor of a software component handling a specific workload.

- *Scaling Precision*: It computes the factor of the scaling of a service process that was successful and according to the agreed SLOs, by dividing the number of successful scaling actions by the total number of scaling actions.

3.2.4 Elasticity Quality Dimension

We have defined **Elasticity** quality dimension as the degree at which a software system is able to autonomously scale the amount software instances or stretch the current VMs exploited (scale up) based on workload fluctuations e.g when the client expects to call a service (possibly multiple times) and gets a result according to the respective quality constraints posed. In order to compute the elasticity of the software components we have defined the following metrics:

- *Mean Time Taken to React (s)*: MTTrct is defined from the moment the need of scaling is detected until the respective scaling is completed. To calculate MTTRct we can define the following metric:
  - *Reaction Time (s)*: Is the raw time that a scaling procedure lasts. A possible example for the calculation of the reaction time could be the time between the request of an additional software component and the instant of actually being up and running.
- *PerfScaleFactor*: Is the scale factor of the performance between two invocations of the same service before and after the scaling has been performed; equals to $\frac{\sum_{i=1}^{N} perfscalefactor_i}{N}$, where *perfscalefactor*$_i$ is the scale factor of i metric and N is the number of metrics being considered.

3.2.5 Security Quality Dimension

As defined in Section 3.1.4 of the Workflow Model we can define the **Security** Dimension in favor of a SQM as the dimension describing means of providing *AAA(auditability, authorisability and authenticity), confidentiality, and non-reputability* [25]:

- We have used the notions of *Incident Detection* and *Authentication*. These notions describe the degree at which services are being detected with security incidents and a more specialized occasion where we calculate improper authentication upon the usage of services(being also a incident detection). These metrics are:
  - *Authentication Security Rate*: This metric depicts the rate at which the service is safe of authentication intrusions.
  - *Incident Detection Rate*: This metric depicts the rate at which the service is suffered from security incidents.
  - *Mean Time to Incident Recovery (s)*: Mean time to incident discovery [25] indicates the effectiveness of software components in the detection of incidents, by measuring the average elapsed time between the initial occurrence of an incident and the discovery thereof.
    - It is calculated by the division of amount of time, in hours, that elapsed between the Date of Occurrence and the Date of Discovery for a given set of incidents, divided by the number of incidents. MTTID equals to $\frac{\sum_{i=1}^{N} DateOfDiscovery_i - DateOfOccurence}{NumberOfIncidents}$

Security Raw metric	Security Composite metric
Date of Discovery of an incident	MTTID
Date of Occurrence of an incident	MTTID
Number of Incidents	MTTID
Number of Software Components without known severe vulnerabilities	SoftWithoutVulner
Number of scanned software components	SoftWithoutVulner

Table 3.7: Match between raw and composite metrics based on the calculation of the composite metrics

Security Composite metrics	Calculation Formulas
Mean Time to Incident Recovery	$\frac{\sum_{i=1}^{N} DateOfDiscovery_i - DateOfOccurence}{NumberOfIncidents}$
Percent of Software Components without known severe vulnerabilities	$\frac{SoftComponentsWithoutSevereVulner}{ScannedSoftComponents} * 100$

Table 3.8: Composite metrics calculation formulas

– *Percent of Software Components without known severe vulnerabilities (%)(SoftWithoutVulner)*: This metric measures software components's exposure to known severe vulnerabilities. It evaluates the percentage of software components being scanned that do not have any known high [25] severity vulnerabilities. The calculation of it is based on the number of software components without known severe vulnerabilities divided by the number of scanned software components.

To compute the above metrics we define the same calculation metrics as the ones defined in section 3.1.4 of the Task Incident Rate and Task Authentication, Authentication Security Rate and and Incident Detection Rate security metrics, whenever the task represents the functionality of the according service component. We represent two tables, where in the Table 3.7 there is a list of the of raw security metrics and their according composite metrics that are being used. In Table 3.8we represent the composite security metrics and their calculation formulas.

3.3 Infrastructure Quality Model

Relying in [13] we have used five quality attributes and one quality dimension. Mainly PaaS, Network-as-a-Service (NaaS) and IaaS are the reference points of most of the quality attributes that are being defined, as also for the metrics that are being defined for each one of them.

3.3.1 Performance Quality attribute

Performance quality attribute is being used in order to characterize the underlying cloud's infrastructure performance in terms of response times upon actions of VM deployment/re-deployment, software deployment/re-deployment, migration of service instances between different VMs and replication of software components between types of different VMs. Metrics that characterize the above procedures are the following:

- *Deployment Time (s)*: Time needed in order to deploy a service component
- *Redeployment Time (s)*: Time needed in order to re-deploy an already deployment service instance on the same VM.
- *Replication Time (s)*: Time needed in order to replicate a service component from one VM to another.
- *Migration Time (s)*: Migration time is the time needed in order a software component to be transferred from one VM to a different one.

The same metrics as above are also applied on two different situations:

1. The first relates the above metrics for the shake of VMs that function within a single and unique cloud infrastructure
2. And the second occasion comes when the above metrics refer to the actions of replication and migration of a VM but this time between different cloud providers.

3.3.2 Networking Quality attribute

Networking quality attribute characterises the quality of network between a data center's SaaS/Application provider and the client of SaaS . In order to assess internally the network performance, we cover application/SaaS view of the according data center and also of it's client using SaaS. We characterize the network quality attribute, when an application execution involves the invocation of different components, which matters for the application/SaaS provider view and it's client side also. Such metrics are:

- *Mean Packet Loss Frequency (lost packets/min)*: indicates the mean rate of lost packets that failed to arrive at their destination
- *Max Connection Error Rate*: Defines the maximum rate at which connection errors occur.
- *Response Time (s)*: Indicates the time where the SaaS reacts upon client requests, including the networking delay.
- *Packet Round-Trip Transfer Time (s)*: Indicates SaaS/Application provider view of the mean packet transfer time from/to it's source/destination accordingly.

3.3.3 Bandwidth Quality attribute

Bandwidth quality attribute characterizes the volume of information per unit of time that a transmission medium (e.g., I/O device, Network Interface Controller) can handle. Thus we have defined the following metrics:

- *Max Incoming Bandwidth (megabits per second (Mb/s))*: Metric of maximum incoming bandwidth is defined as the maximum rate of Mb/s per second of the respective transmission medium, induced from the raw incoming bandwidth measurements.
- *Min Incoming Bandwidth (megabits per second (Mb/s))*: Indicates the minimum incoming bandwidth rate of Mb/s per second of the respective transmission medium.

We follow the same definitions in favour of outgoing bandwidth with *max* and *min* values.

3.3.4 CPU Utilization Quality attribute

Quality attribute of *CPU Utilization* depicts the level at which processors are being leveraged within a cloud infrastructure in favour of a client under the two different types of hypervisors. Some of the metrics that could be measured are:

- *Type of hypervisor*: This metric retrieves the type of the hypervisor classified as stated in [26]. vCPUs are not being spawned in a uniform way thus we can define the timestamp metric of full schedule CPU where the hypervisor schedules all the vCPUs that are being demanded. Having these information we can refer to metrics of the arrival rate and CPU Average Load as to determine the point of load that the vCPU is being spawned.
- *CPU Usage (%)*: This metric depicts the measure of the percentage of CPU/s (in a multi-CPU hardware architecture) cycles dedicated upon running processes. Outputs the individual values of usage for each of the CPUs and a global usage rate among them.
- *Shared Physical CPUs*: Indicating the number of different VMs that the physical CPU is being used of.
- *Network shared Physical CPUs*: Indicating the number of different VMs that the physical CPU is being used of but within different network clusters
- *Virtual CPUs*: This metric depicts the number of Virtual CPUs that are being served by each of the physical CPUs and scheduled by the according hypervisor
- *CPU Average Load (%)*: Indicating the dynamically changed value of the average CPU load for a processor
- *Multi-CPU Overall Maximum Load (%)*: This metric indicates the percentage of maximum Load over all processors in a multi-processor architecture.

  - *CPU Maximum Load (%)*: Indicating the peak load of the CPU within a specific time period of reference

- *CPU Overall Minimum Load (%)*: This metrics indicate the overall minimum CPU Load that has been monitored on each of the processors. Calculation of it includes the use of the following metric

  - *CPU Minimum Load (%)*:Indicating the minimum load of the CPU within a specific unit time of interest

- *Multi-CPU Overall Minimum Load (%)*: This metric indicates the percentage of minimum Load over all processors in a multi-processor architecture.

3.3.5 Memory and Storage Quality attributes

From the position of accessing and storing data we have defined metrics for the quality attributes of *memory* and *storage* for the following types: RAM and $Hard - Disks$. Metrics to assess the quality of storage for RAM are the following:

- *RAM Access Time (%)*: Percentage of time the RAM is being accessed .

- *Maximum RAM bit/Byte Speed (MB/s)*: Indicates the maximum RAM speed via which data are being transferred (from/to the RAM)

- *Minimum RAM bit/Byte Speed (MB/s)*: Indicates the minimum RAM speed as transfers of data in MBs per second

- *Average RAM Utilization (%)*: This metrics depicts the average utilization of RAM for a certain period of time. We calculation the average utilization of RAM cause RAM often fluctuates between very short period of times (e.g between one or two seconds we can have major fluctuations)

Quality of storage of Hard-Disk is mainly dedicated to the performance of slow-accessing hard disk drives in case of storing and retrieving data:

- *Mean Hard Disk Write Speed (KB/s)*: Defines the mean Hard Disk write speed being recorded within a certain unit time of interest

- *Maximum Hard Disk Write Speed (KB/s)*: Defines the maximum Hard Disk write speed being recorded within a certain unit time of interest

- *Minimum Hard Disk Write Speed (KB/s)*: Defines the minimum Hard Disk write speed being recorded within a certain unit time of interest

- *Hard Disk Utilization (%)*: This metrics defines the utilization of the Hard Disk for a certain point of time.

Metrics applied for the procedure of *Write* could also be applied for the procedure of *Read* for a Hard-Disk.

3.3.6 PaaS/IaaS Scalability and Elasticity Quality attributes

Based on [15] we define PaaS/IaaS *Scalability* as the ability of the underlying infrastructure to sustain increasing workloads by making use of additional resources, which are directly requested, including all the hardware and virtualization layers. Metrics for the IaaS scalability quality attribute could be:

- *Scalability Range (ScR req/min)*: Based on [14] we have defined ScR as a scalability metric that reflects the ability to handle maximum workload that can still be tolerated by the underlying infrastructure to still satisfy the SLOs posed to it as long as the infrastructure resources limitations permits it, depending on the resource type that is being demanded

- *Scalability Speed (ScS req/min)*: Based on [14] we define scalability speed as the speed that the underlying cloud infrastructure can handle after the scaling point in order to achieve it's relative SLOs.

Elasticity [15] depicts the degree to which a cloud infrastructure is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner such that at each point in time the available resources match the according demand, based on user requirements, as closely as possible. In addition, in order to describe metrics of elasticity we depend on the two core aspects of *precision* and speed. Precision of scaling [15] is defined as the absolute deviation of the according amount of allocated resources from the actual resource demand. Based on the aforementioned aspects of elasticity, metrics that are being defined [15] are:

- *Average Scaling out Time (Asc-out)*: Metric of Asc-out is defined as the average time to switch from an underprovisioned state to an optimal overprovisioned state (excluding the cases where the demand is accurately matched) corresponding to the average speed of scaling out

- *Precision of scaling out (P_O)*: P_O is defined as the average precision depicting the accuracy of the scaling out actions being performed that had a satisfactory result.

- *Elastic Scaling out (E_O)*: Factor of E_O indicates the degree that can be achieved from the scaling out action

- *Asc-in, P_I , E_I*: Those metrics are being defined accordingly as the previous ones but this time caring for the *scaling In* procedure.

- *Mean Time To Quality Repair (MTTQR)(min)*: MTTQR defines the mean time a IaaS needs to satisfy its specific SLOs (dedicated to IaaS) for a specified workload for the time period between a starting scaling point and an ending scaling point. In order to calculate MTTQR, we have defined also the raw metric of:

  - *TimeToQualityRepair(TTQR)(milli)*: Metric of TTQR defines the difference in time between two time points. These time points indicate the max rate/minutes before the scaling action and the max req/min after the scaling action has

Scalability/Elasticity Metrics	Calculation Formulas
I	average amount of underprovisioned resources during an underprovisioned period with respect to SLOs
O	average amount of overprovisioned resources during an overprovisioned period with respects to SLOs
P_O	$\frac{\sum_O}{T}$
P_I	$\frac{\sum_I}{T}$
E_O	$\frac{1}{Asc-out}$ * O
E_I	$\frac{1}{Asc-in}$ * I
ScR	max value of req/min
ScS	(max,rate)

Table 3.9: Scalability/Elasticity metric calculation formulas

ended. By subtracting the time point before the scaling from the time point after the scaling, we can calculate TTQR for the the time period of interest where SLOs are possed on to it.

- *Number of SLO violations (NSLOV)*: NSLOV reflects the number of SLO violations after a specific scaling time period when workload changes at a given rate, being measured as a real number. For example, with a workload increase factor of 1.4, a perfectly elastic system would have 0 SLO violations per request, i.e., NSLOVIaaS(1.4 req/s) = 0. On the other hand, a less elastic system will violate one SLO for the time between the scaling points, i.e., NSLOVIaaS (1.4 req/s) = 1 in this case

Table 3.9 indicates the calculation formulas of elasticity/scalability metrics. Symbol *T* indicates the total duration of evaluation period.

3.3.7 Security Quality Dimension

As far as concerning the **Security** dimension the same metrics apply here as depicted in the Service Quality Model that follows on the upcoming section. The differentiation here is that we consider system components like VMs instead of service components in order to calculate metrics that describe the quality attributes of authentication, confidentiality, auditability as being defined in the SQM.

3.4 Cross-Layer Dependency Quality Model

To formalize relationships between quality metrics across the three layer-specific QMs, we have considered an initial set of cross-layer dependencies in the form of a **Dependency Quality Model**. These dependencies indicate that (a) the computation of a metric on layer *X* can be used on layer *X+1* to complete the computation (or part of it) of a relevant metric

and (b) metric values from different layers that are inter-dependent, for example the high or low value of a metric in layer *X* can affect the value of a metric in layer *X+1*. Relevance could map to either both metrics belonging to the same quality dimension, or being described by similar quality attributes. Via the cross-layer dependencies (Figure 1.1), the metric aggregation formulas and the fact that raw quality metrics with no dependencies can be calculated by sensors placed by the distributed monitoring system on one of the respective layers, the measurability of all metrics defined is guaranteed. We have separated the cross-layer dependencies between the metrics by considering groups of adjacent layer-specific QMs. For the *SM* and *WM* group the following dependencies have been captured:

- *Task execution time* of a service task in a workflow can be computed from the *execution time of the service* used to realise this task's functionality.
- *Service task availability* equals the *availability of the service* used to realise the task's functionality.
- The computation of *task's fidelity* can be used in order to describe the *fidelity of the service* realising its functionality.
- Metric of *Entity Delay Time* in *WM* can be used in order to derive the value of the *Delay Time* of a service component defined in *SM*.
- We derive the value of the *Task Failure Time* defined in *WM* by computation of *1-Successability* metric defined in *SM*. By this computation we remove the successful request in order to result on the rate at which the task was in failure state.
- Metric defined *WM* concerning the *authentication and accessing* on task's are equal with the computations of *authentication and confidentiality* defined in *SM* when internal services are being monitored.

Next group of dependencies derived are from the *IM* and *SM* QMs:

- A metric we use from *IM* is the *Mean Time to Quality Repair*, which has an *equality* reference to *SM* for the *Mean Time Taken To React* defined in *elasticity dimension*. Based on the definitions of the two metrics, they can be cross-referenced whenever the Mean Time Taken To React refers to scaling up actions of the IaaS component that scaled in order to satisfy its specific SLOs for a specific workload.
- There is an increasing trend that relates *Scaling Utilization* defined in *SM* and *Scaling Range* defined in *IM*. When utilization of scaling is high that means that the underlying infrastructure is capable of handling scaling actions in high rates, meaning that the scaling range has also an increasing value making them inter-dependent values.
- *Adaptation time* of *SM* is correlated with *scalability speed* defined in *IM* when actions of a service component being adapted had as a result the scaling of the

underlying infrastructure this way the adaptation time equals with the scalability speed or scalability speed is part of the adaptation for the service component.

- *Precision of Adaptation* of a service component can be derived from the *Precision of Scaling* defined in *IM* when adaptation and scaling occurs for the same virtualized infrastructure component and it's SLOs.

- Regarding the *CPU utilization* defined in *IM* we can infer that is an inter-dependent value with the *response time* defined in *SM*. CPU utilization is increased if the overheads associated with context switching are being minimized and happen infrequently. This will have as a result the increase of the execution time for the according services.

- In order to compute *Scaling Precision* defined in *SM* we must first calculate the *Number of SLO violations* defined in *IM* in order to calculate the precision of scaling having known the number of scaling actions that failed due to SLO mismatches.

Chapter 4

Monitoring System Architecture

This chapter demonstrates the logical and physical architecture of the monitoring system, as also the type of the ManagementDB that was selected in order to store the data of the quality metrics.

4.1 Logical Architecture

In order to support the QMs presented in the Chapter 3 we have developed a distributed cross-layer monitoring system along with the use an adaptation system developed in [1]. Main idea of our approach is that measurements at different layers are encapsulated by sensors attached to respective layer-specific components and that these measurements are stored in Management Databases (Section 4.2.1). Figure 4.1 depicts the logical architecture of our distributed monitoring framework along with the support of the adaptation system. We are mainly focused on the left part of Figure 4.1 where the following actions are taking place:

1. gathering monitoring data
2. aggregate/assess data and
3. storing them in the according management database

Architecture of the the distributed monitoring framework consists of three Monitoring Managers for the Workflow, Service and Infrastructure layers. In each of these layers a metric aggregator and an assessment component is being defined which are responsible for (a) collecting and calculating metric measurements based on the according QM and (b) passing them to the assessment component which is responsible for the detection of specific SLO violations for the according layer. For each of the three different Monitoring Managers of the according layer we have defined the following functionalities:

- Service and Infrastructure Monitoring managers support three kind of basic actions
  - Aggregation of information based on the quality metrics that we defined in each of the layers.

  - Assessment of the aggregated information in order to detect SLO violations
  - Storing of aggregated data to the management database. According to the level of the aggregation being supported where in this case we selected to store metric data in TSBD instances (ref 4.2.1).
  - As the cross-layer dependencies have been spotted, Service and Infrastructure managers propagates the measurement in the instance of WM in order to fill possible metric values that where identified during the cross-layer dependency done. The propagation is done from lower to higher levels e.g, SM to WM.

- Workflow Monitoring manager besides the three actions of aggregation, assessment and storing of metric data in ManagementDB, it also supports:

  - The functionality of a cloud dependency aggregator which actually implements the cross-layer dependency model that has been defined. We have added this required functionality on the Workflow Monitor Manager in order to expose cross-layer dependencies detected within the same public cloud sector.

To support multi-cloud dependencies as depicted in Figure 4.1 we publish metrics measurements that might have cross-layer dependencies with metrics that are being taken from different infrastructure cloud sectors. The Global Cross-Layer Dependency Metrics Aggregator component is different from the according central aggregation component. The required inputs for the Cross-Layer Dependency Metrics Aggregator are:

- Metrics coming from the three layers that could be part of a cross-layer dependency relation.
- Aggregated information indicating the cross-layer dependencies that have already been defined for the according public sector along with the applied multi-cloud SLOs.

Processing the above inputs, the centralized global component serves the following functionalities:

- Publish the needed adaptation actions, in case of SLOs violations, on the adaptation framework.
- Infers the detected patterns of monitoring data to the Patter Discoverer implemented in [1]
- Aggregates metrics in case of cross-layer dependencies from different cloud providers.
- Stores global aggregated information in a ManagementDB instance.

These functionalities of the centralized components has the advantage that, in case of failure of one of the monitoring managers or the centralized aggregator, the according cross-layer dependency information could be retrieved or recalculated. In the end we state that we also support a database where SLOs, Event Models and Adaptation Models are being stored and passed on the component of adaptation and monitoring system.

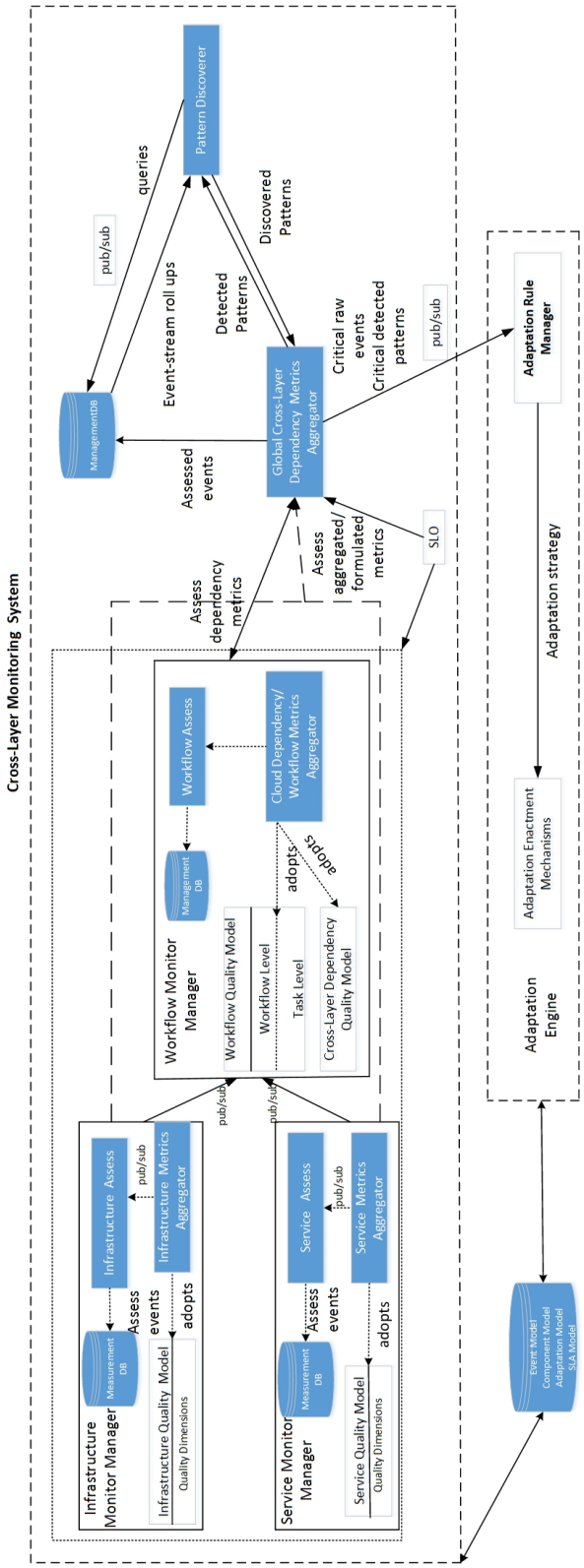


Figure 4.1: Monitoring System Logical architecture

4.2 Physical Architecture

4.2.1 Time Series and Event Processing Databases Comparison

We can categorize monitoring data into (a) *metric based data* and (b) *event based data*. The implementation and theoretical background of this research is referring to the use of a metric based system, based on the definition of each of the QM metrics.

The differentiation between event based and metrics based systems has to do with the layers that are being defined in order to be monitored. In this thesis we have spotted the difference between a metric-based and an event based monitoring system on the fact that event based monitoring systems are more practical upon metrics that have a business value (e.g use of a KPI, Business layer). To be more precise, let's state the situation where there is a KPI indicating the fact that whenever a credit card exceeds a certain money boundary then the customer has to be informed. If we had to deal with such a calculation then the event-based data and Event Processing databases would be the most suitable choice for two reasons, (a) the fact that we care about real time event data is a major advantage of Event Processing databases and (b) the fact that a business value is being monitored and not a metric that deals with structural and raw data information.

On the contrary, based on our definitions for the QMs we conclude on the use of Time Series Databases(TSDB) [27] in order to support our metric based QMs which do not govern the notion of business values and KPIs, but instead they consider raw values of metrics that could represent counters, seconds and summaries based on structural and component monitoring data. Structural monitoring data could refer for example on the workflow layer where the sequence path of the workflow with the according monitoring metrics that supports it, is concerned. Component monitoring data could refer to metrics defined the SM layer and concerns the response times of the according software component. Thus, in such type of QMs we care about storing these values on the according TSDB along with a timestamp of their detection. In the next section we demonstrate our architecture and refer to usage of the according databases.

4.2.2 Cloud Infrastructure

The physical architecture of our distributed monitoring system consists of *N* hybrid cloud providers and a public cloud domain which is represented by one VM. We define hybrid cloud as the cloud infrastructure that consists of public and/or private infrastructure components. VMs are being provisioned/de-provisioned by the PaaS tool which is capable of monitoring specific Infrastructure layer quality(for this functionality we have used a PaaS simulator as depicted in section 6.3). Each of the user VMs in each of the hybrid providers include (a) monitoring sensors, (b) aggregators/assessment components and (c) database instances.

Based on Figure 4.2 we have more than one VMs that compromise the Infrastructure and Service Monitor Managers, cause of the distributed nature of the monitoring system. Action of assessment is also done the component of the aggregation, that is the reason seen as one component. Information produced by the user VMs is being passed on the WF En-

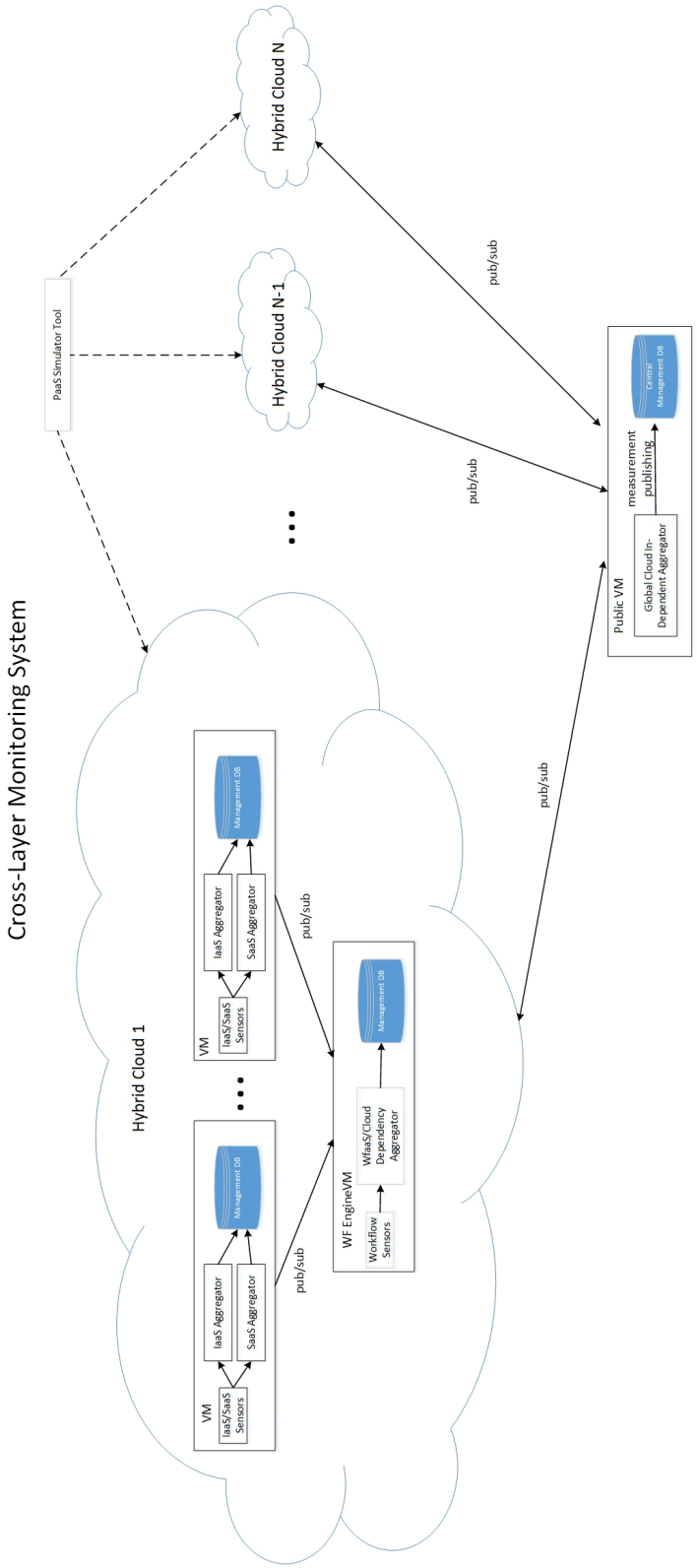


Figure 4.2: Monitoring System Physical architecture

gine VM by a publish/subscribing system, which could be either a functionality offered by the monitoring tools that are going to be used or a custom implementation of producer/consumer communication. The role of this VM is not only to have a WfaaS aggregator but also a cloud-dependent aggregator which will aggregate cross-layer dependency metric values whenever applicable. Workflow engine is a cloud-centralized component and that is the reason that only one user VM that includes the according WF Engine is being provisioned on the hybrid cloud sector. Cross-layer dependency information is being derived at run time and is delivered to the according user VM in favor of measurement propagation whenever cross-layer dependencies between VMs are applicable.

Global Cloud-independent Aggregator VM and TSDB management database are public infrastructure components. Responsibility of the VM is to aggregate cross-layer dependency values obtained from different cloud providers. In contrast to the WF Engine/Dependecy aggregators which take in account only specific cloud metric aggregations. Figure 4.2 represents in a double single arrow the metric measurement information passing from the cloud to the public cloud infrastructure. In essence this arrow represents:

- Metric measurement information from each of the three different user VMs, publishing specific aggregated layer information to the public cloud infrastructure
- Propagation of metric measurements according to the cross-layer dependencies identified within the Hybrid Cloud.

On the next step the information is being passed on the adaptation and pattern discovery systems [1] which are out of the scope of the physical architecture.

Chapter 5

Monitoring System Implementation

In order to implement the proposed distributed monitoring system of this thesis, we have used and tested several monitoring tools and picked the ones that are suitable for our needs. Most of the reviewed tools are spotted at the Service layer. Concerning the Infrastructure layer, monitoring tools are much alike and no need for comparison was needed as we ended in straight forward solution. For the Workflow level have followed the solution that Activiti [18] provided in order to monitor the according workflows. In the following sections first, we make a comparison of the monitoring tools and advocate our on choice and then we depict the implementation of the the monitoring sensors that QMs support.

5.1 Comparison of Service Monitoring tools

Most of the monitoring tools have pricing policies but there also freeware solutions that can be used. Our comparison is based on a spectrum of freeware and open sources monitoring tools, thus we are going to compare the free offers of each of the monitoring tools that supports a pricing model along with the open source monitoring solutions.

Table 5.1compares service monitoring tools, based on a set of criteria. All of the mentioned monitoring tools support metrics in order to measure i) response times ii) throughput and iii) availability, service metrics. Thus, we mainly focus on the following criteria in order to evaluate and choose the monitoring tool that will serve us: a) the chance of being open source b) the local domain support, which represents the possibility of the monitoring tools offering monitoring capabilities in a localhost domain c) the fact of not having an upper limit of the monitoring events that are going to be recorded d) definition of custom metrics in order to measure specific software components e) support of Real User Monitoring (RUM) [28] which is a major factor of computing response times and finally f) standalone solution as we cater for monitoring tools that do not depend on third party back-end components. The last aforementioned criterion is chosen in order to value the simplicity of the monitoring system that we are going to support.

Based on Table 5.1, *Local Domain Support* is applied by all the monitoring tools that we have selected to compare by the means that a particular user can have her services functioning in sub-domain or localhost and still be monitored by the according tools. An

Criteria / Monitor. tool	**Open source**	**Local domain Support**	**Unlimited Storing**	**Custom Metrics**	**Support of Real User Monitoring**	**Standalone**
Datadog, [29]		✓		✓		✓
New Relic, [30]		✓				✓
Prometheus, [31]	✓	✓	✓	✓	✓	✓
Telegraph (Influxdata), [32]	✓	✓	✓			
statsD (Graphite), [33]	✓	✓	✓	✓	✓	
scollector (Bosun), [34]	✓	✓	✓	✓		

Table 5.1: Service Monitoring tools comparison

upper-limit on storing data metrics is only met by monitoring tools that comes with a pricing model. To be more precise *Datadog* [29] monitoring tool offers 1 day of retention of the data metrics along with the definition of custom metrics and NewRelic [30] gives a trial period of 30 days and then supports only basic and custom metrics. All of the rest monitoring tools, as being open-source, do not have an upper limit on the storing events. A major factor in the monitoring functionality of the Service layer (as we discuss in the following section) is the fact of defining custom metrics in order to monitor specific software components. Support of custom metrics is not supported by New Relic ,as it is a commercial tool, and telegraph tool which offers specific monitoring aspects mostly for the Infrastructure layer (CPU throughput, memory, etc). For the rest of the tools we are free to define custom metrics in order to measure specific metrics of the software components that we have defined in SM. One major aspect for the calculation of the response times in real time services is the *Real User Monitoring (RUM)* which is mainly supported by statsD [33]. With the support of RUM we can retrieve specific measurements in order to calculate metrics of service response and execution times by adapting custom metrics in the backend of software components. Finally, the criterion of *StandAlone* indicates the fact that a monitoring tool does not support third-party software solutions and can be given as a unified monitoring component. This criterion is adopted by the commercial monitoring tools (Datadog and NewRelic) and by Prometheus [31] open-source monitoring solution. The rest of the tools depend on third-party servers in order to function and collect data metrics(e.g statsD on Graphite [33] and scollector on Bosum server [34]).

As we have described the criteria and their support by each of the monitoring tools we ended up in the selection of Prometheus in order to retrieve data metrics for the SM layer. The main reasons for this selection are a) It is an open-source monitoring solution that leverages the custom creation of metrics in a way that fits most for the implementation that we have followed (described in the below section) and b) it can be viewed as a unified monitoring component that does not depend on *third-party monitoring* components decreasing this way the possible complexity that multi-depend monitoring software could

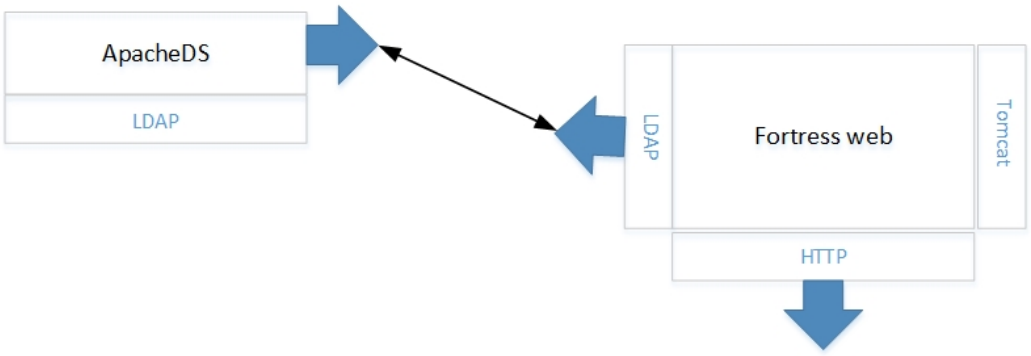


Figure 5.1: Communication between ApacheDS and Fortress-Web

offer. Additionally we have also extended Prometheus in order to support RUM with the addition of a proper front-end, third-party [35], *javascript library*.

In the following section we describe how we use Prometheus in our monitoring framework to monitor the Service layer. Prometheus is being used in such a way as to retrieve raw metrics that will serve in the aggregation procedures of the metrics defined in SM.

5.2 Service Layer Monitoring sensors

5.2.1 Service-based project of Fortress-Web

In order to implement the SM layer we have decided to proceed with the selection of a group of services which constitute the project of Apache-Fortress [36]. We have selected this project because it provides a variety of service components that we can monitor in order to fulfill our needs (adaptation of SM layer). Apache-fortress web is an open source project which is licensed under the Apache foundation projects [37] and the purpose of it is the support of a LDAP [38] directory that includes a number of users and groups that can be used in order to give access on third party applications. More precisely it is an access management system, that provides role-based access control (RBAC) [39], delegated adminstration and password policy services being served by the LDAP implementation of it. Fortress-web has the following architecture dependencies:

- An apache directory server (ApacheDS) [40] that implements the functionality of the LDAP server.
- And an apache tomcat [41] web server that provides the functionality of manipulating the content of the LDAP server. The end user uses HTTP in order to have access on the according services that web server offers.

Fortress web communicates with ApacheDS, by exchanging information whenever actions(retrieval of information, creation, deletion and edition) are being done upon entries of ApacheDS.

A number of 11 services which we are separately monitoring, constitutes fortress-web. These services are much alike between them as they give the opportunity to the end-user to create, delete and edit users/groups stored in the LDAP server. Main reason of selecting this open source project is that we have adapted a monitoring pattern in each of these services (explain below) that can provide us with a clean overview of the actions that an end-user might take in order to retrieve data for metrics defined in SM. Furthermore, from a technical perspective, apache-fortress is written in Java which is fully supported by the monitoring tool that we have selected (Prometheus) providing us this way a straight-forwarded solution in the monitoring procedure.

5.2.2 Prometheus Monitoring solution

As we have already stated in the comparison section 5.1, Prometheus software monitoring solution fits well to our needs for five main reasons:

- It is an open source project
- It's architecture does not depend on third party monitoring tools
- We can use prometheus in order to record any purely time series metrics which we store in the according managementDB (Section 4.2.1)
- It is applicable upon highly dynamic service-oriented architectures (as the one of fortress web project) and
- We can adapt our on monitoring needs by defining metrics that implement the Service Quality Model.

A major advantage of prometheus is the perception of *monitoring orchestration*. Monitoring orchestration is a procedure in which a developer of a web service adapts her monitoring needs by defining metrics in order to monitor aspects that deals with the service component per se. Prometheus follows this direction by giving the chance to the developer to orchestrate her service component of interest, based on metrics that she has defined (e.g metric that depicts the number of access for a certain service component). Prometheus then automates the procedure of retrieval of this kind of metrics information in real-time by scraping the data and storing them on a local storage subsystem.

Figure 5.2 depicts the Prometheus four-stage architecture:

1. In the first stage, orchestration of the according services is a necessary procedure in order to retrieve raw values of metrics(see example below)
2. In the next step we support the usage of Prometheus exporters [42] which are custom collectors available as open-source projects that have already pre-defined metrics adaptable to our needs

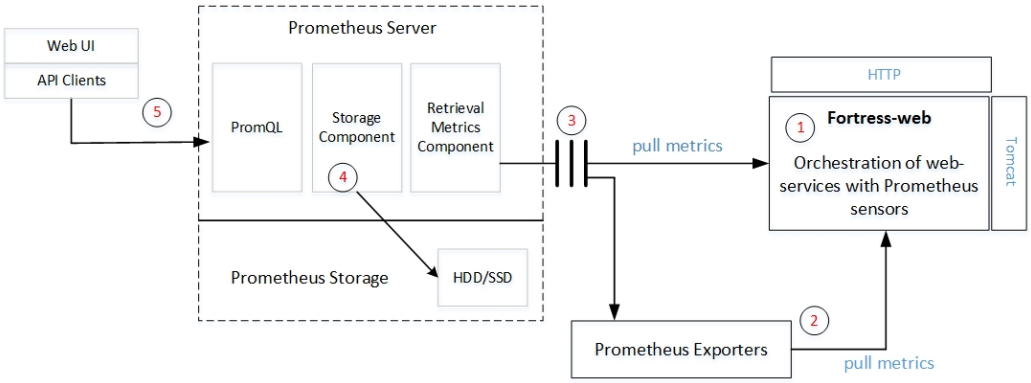


Figure 5.2: Prometheus Monitoring architecture

3. In the third stage Prometheus server pulls metric data from all the available sources, which could be orchestrated software components and prometheus exporters that have already pulled the appropriate metric data from the monitored software components.

4. in the fourth stage it actually stores the data in a hard disk storage.

Prometheus client libraries offer four core metric types. These are differentiated in the client libraries in order to enable APIs tailored to the usage of the specific types. We mainly support two out of them which are a) *counters* and b) *summary* metrics. A counter metric type is a cumulative type of metric that represents a single numerical value that only ever goes up. Type of counter metrics are typically used to count requests served, tasks completed, errors occurred, etc. Counters are not used to expose current counts of items whose number can also go down, e.g. the number of currently running co-routines. On the other hand summary metric types sample observations (like request durations and response sizes). While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over specific time periods.

In order to explain the way that we orchestrated the monitoring functionality and the definition of metrics, we demonstrate the usage of *Counter* and *Summary* metrics type done in the first step of the monitoring procedure.

```
1  import io.prometheus.client.Counter;
2  import io.prometheus.client.Summary;
3
4  class SoftwareComponent {
5    static final Summary userProcessTimeLatency = Summary.build()
6        .name("userProcess_latency_milliseconds")
7        .help("User Process Time milliseconds").register();
8    static final Counter userRequestNumber = Counter.build()
9        .name("user_request_total")
10       .help("User Request Total").register();
11
12   void UserPage() {
13     Summary.Timer requestTimer = userProcessTimeLatency.startTimer();
14     try {
15       // service component implementation code.
```

```
16      } catch (Exception e) {
17        throw e;
18      } finally {
19        requestTimer.observeDuration();
20        userRequestNumber.inc() ;
21      }
22    }
23  }
```

Listing 5.1: Counter and Summary metrics orchestration

In Listing 5.1 we have defined a Counter and a Summary metric. Counter metric calculates the times that the according service component has been requested and the Summary metric calculates the process time being taken in order for the request to be executed. Features of the orchestration depicts the following:

- Each metric has a unique name, specified with the `name()` method.
- Each metric has a description, specified with the `help()` method.
- Metrics are usually constructed only once, the orchestration client is threadsafe.
- Orchestration is an integral part of the code, making it easy to understand and maintain.

Once we've orchestrated our software component, then this information is being exposed to Prometheus by a HTTP servlet that we have added on fortress-web. Following this kind of orchestration we are capable of retrieving raw metric data information being stored on the fourth stage of architecture and accessible by HTTP or a Web UI(offered by Prometheus) requests in the fifth stage as Figure 5.2 shows. We explain the procedure of retrieval of metrics in Section 6.1 were we demonstrate the implementation and functionality of the aggregators and more precisely the SaaS aggregator component. Aggregation procedure follows three steps a) retrieval of metrics based on certain API calls b) aggregation procedures in order to calculate composite metrics and c) storing of the results in a TSDB instance.

5.3 Infrastructure Layer Monitoring sensors

Nagios [43] monitoring tool is used in order to implement various infrastructure monitoring sensors. We have used Nagios Core as it is the Open Source solution of system and networking monitoring that Nagios Enterprise offers. Nagios Core can monitor hosts and infrastructure functionalities, alerting when procedures are in caution state and when they are in accepted state. Nagios Core is mainly designed to run under Linux distribution and the main reasons that we have used it is that (a) it's an open source solution ,(b) it's highly configurable as we can define our own on Nagios plug-ins and (c) it's a distributed component monitoring solution that fits upon any infrastructure physical architecture.

Figure 5.3 presents the monitoring procedure of each of the Infrastructure sensors/aggregations components which also depicts the architecture of the Infrastructure sensor/aggregation implementation. Placement and retrieval of raw data is done in the first step

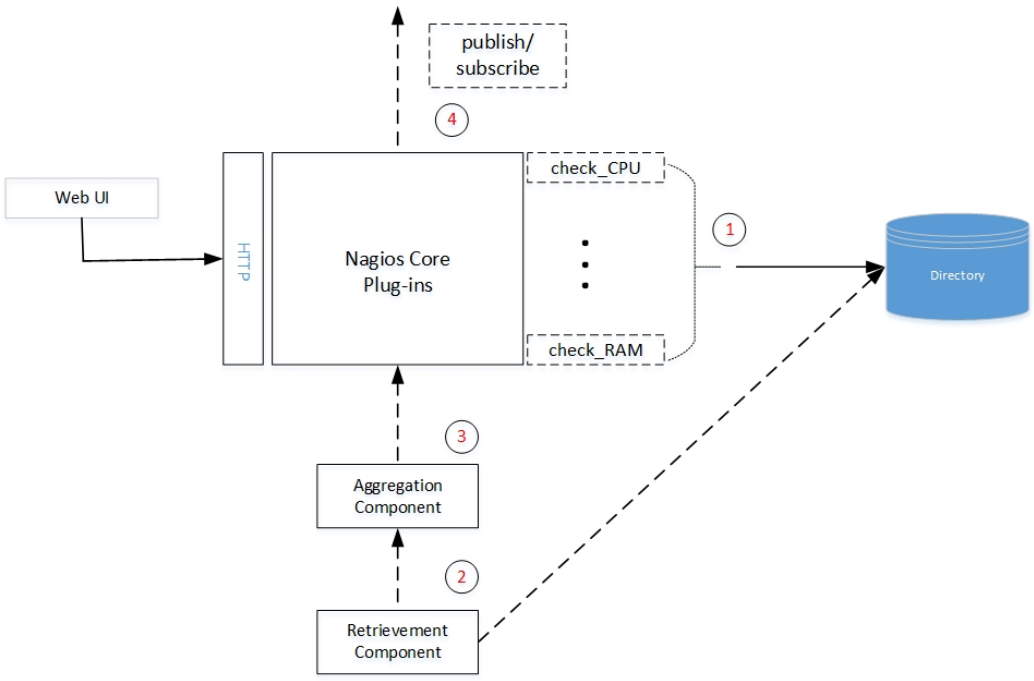


Figure 5.3: Nagios Monitoring architecture

of the IaaS monitoring procedure. At this step we have relied on the component of Nagios Core, plugins available in Nagios Exchange community [44] along with plugins of our own implementation in order to retrieve raw data for the purposes of (a) raw data metrics and (b) calculation of composite metrics defined in IM. Those kind of metrics are represented by the prefix of $check\_metricName$ which depicts the retrieval of the raw measurement. Nagios approves two types of retrieval monitoring data which are (a) *active checks* and (b) *passive checks*. Active checks are used in the first step of the procedure. In this implementation we have not used any passive checks, as we are following a pulling/acting procedure to get the according metric data. To be more precise in the first step Nagios generates active checks launched by the Nagios Core component. Each time the monitoring data are retrieved they are written on the storage of a *.dat* file which resembles the directory component on the Figure 5.3. Implementation of each of the checks that retrieve the monitoring data is done in two steps. In the first step in order to register a check we have to modify a specific .cfg file (*/usr/local/nagios/etc/objects/commands.cfg*) with the according new check and in the second step we have to edit the responsible .cfg file (*/usr/local/nagios/etc/objects/localhost.cfg*) that validates which of the active check commands are going to be actually used by Nagios Core. Checks can be written in any scripting language such as Bash, Perl and Python. In this section we dealt with the sensor implementation by referring only to step 1. Rest of the steps are explained in the implementation of the IaaS Aggregator 6.2, where the actual calculation is being done. Descriptively, steps 2 and 3 refer to the aggregation component and how aggregated data metrics are retrieved by the Nagios Core component. Finally, in the last step we refer to

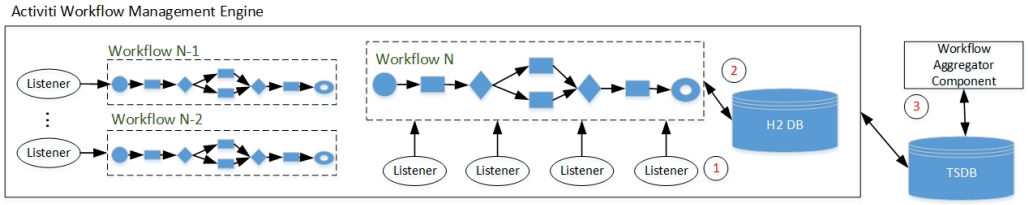


Figure 5.4: Workflow-based Monitoring architecture

the usage of the publish/subscribe system, Section 5.5.

5.4 Workflow Layer Monitoring sensors

At the workflow layer in order to implement the according QM, we have chosen the Activiti Workflow and Business engine [18]. Activiti is an open source Business and Workflow engine framework that provides an environment for running worklfows and business processes. It provides us with functionalities such as the provision of a web-based modeling tool for workflow and business analysts, an Eclipse plug-in for developers which we have mainly used [45] and a web application (.war file) where by using it we were able to manage workflow components. These are the main reasons that drove us to use the Activiti engine in order to fulfil our monitoring needs. Other Workflow engines like Activiti is jBPM [46] and BonitaSoft [47]. The main difference between Activiti and the aforementioned workflow engines is that the activiti project gives the developer a well-structured environment in order to work with making this way a lot easier for the developer to design and sustain their workflows.

The solution that we offer is workflow engine dependent by the means that we cannot adapt the same notion of architecture in other workflow engines. The drawback sight of this fact is that the monitoring solution that we offer is not as abstract as to be used by other workflow engines. Nevertheless, this point only can not minimize the fact that the monitoring implementation is much of importance as we retrieve raw metrics from various quality dimensions as we represent in the according QM.

In Figure 5.4 we represent the architecture of the workflow sensor system which includes N number of workflows where each one of them communicate with an internal database of Activiti (we show only Workflow N in the communication with H2 Database for space saving). The implementation and deployment of monitoring sensors (Listeners) is presented by numbers 1 and 2. We refer to the implementation of the Workflow Transition Delay metric (Section 3.1.1.1) in order to represent the implementation of those two steps. A major entity at the monitoring side of the workflow is the presence of execution listeners [48]. Execution listeners are configurable entities which can monitor the task or workflow and retrieve information of the component being monitored at run time. To return the implementation of the transition delay metric, we have followed two steps, where :

1. In the first step we have attached an execution listener in each of the flows in order

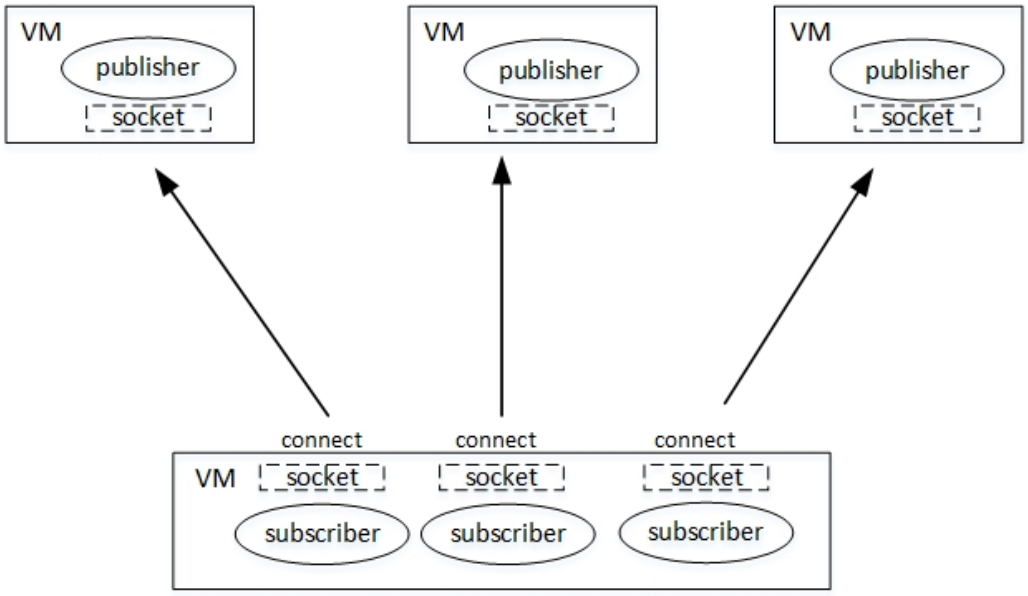


Figure 5.5: Connection between Publishers and Subscribers

to retrieve the actual path that the workflow has followed and at the second step

2. In the last event of the workflow we gather the run time transition delay information and then calculate the transition delays

In the third step, once they are calculated they are being sent to the ManagementDB and then being aggregated by the according aggregation component. Functionality of step three is going to be explained in Section 6.4 where we consider the storage and aggregation points of raw and composite metrics as also the frequency that aggregations are being done. We also want to stress that in case of a workflow/task failure Activiti framework has the functionality of storing the information until the failure point in its internal H2 Database. Thus, if this is the case we are able to retrieve that kind of information by doing simple queries to database where the history of workflow data are actually stored, for example the flow sequence up the point of failure and duration of each of the tasks in the flow sequence of the according workflow.

5.5 Publish/Subscribe system implementation

We have used ZeroMQ library [49] in order to pass he dependency metric measurement calculations from each of the user VMs to the Workflow Engine VM where the dependency aggregator is located, Section 6.5.

As Figure 5.5 shows we have set a publisher in each of the user VMs, which passes with a constant frequency of *five seconds*, the dependency values to the VM that the dependency aggregator is functioning. Responsible for obtaining the dependency values

from each of the publishers endpoints are the subscribers established in the Workflow Engine VM. When these values are successfully received they are passed to the dependency aggregator, Section 6.5 which is responsible to use them in order to calculate the values of the dependency metrics of the according layer. Technically each of the publisher VMs establish a socket in order to publish the metric values. The number of subscribers equals with the number of distinct sockets used for the publishing procedure, thus the same number of separate connections between publishers and subscribers are going to be established, depicting a *one-to-one* relation. For example in Figure 5.5 there are three separate publisher sockets and three subscribers which in order to retrieve the according metric values each one of them connects to a certain (distinct) publisher socket.

5.6 Data Model

Monitoring data values are uniquely identified by their metric (a) *name* and (b) the according *timestamp* indicating the time that the metric has been taken. In order to characterize each of the metrics definition based on certain quality aspects [50], we have used OWL-Q [51] which captures a rich metric model based on the requirements for QoS-based Web Service description.

More precisely as Figure 5.6 depicts, we have used an ontology schema that covers quality aspects such as the types of the quality metrics, attributes, type of their values and the access model that metric data is subject to (pull model). Thus, we are based on this kind of information for each of the quality metrics and quality attributes, in order to describe and retrieve them. In the section of Future Work 8, we are stating our future plans in order to extend and totally attach the use of the OWL-Q schema into our monitoring system implementation.

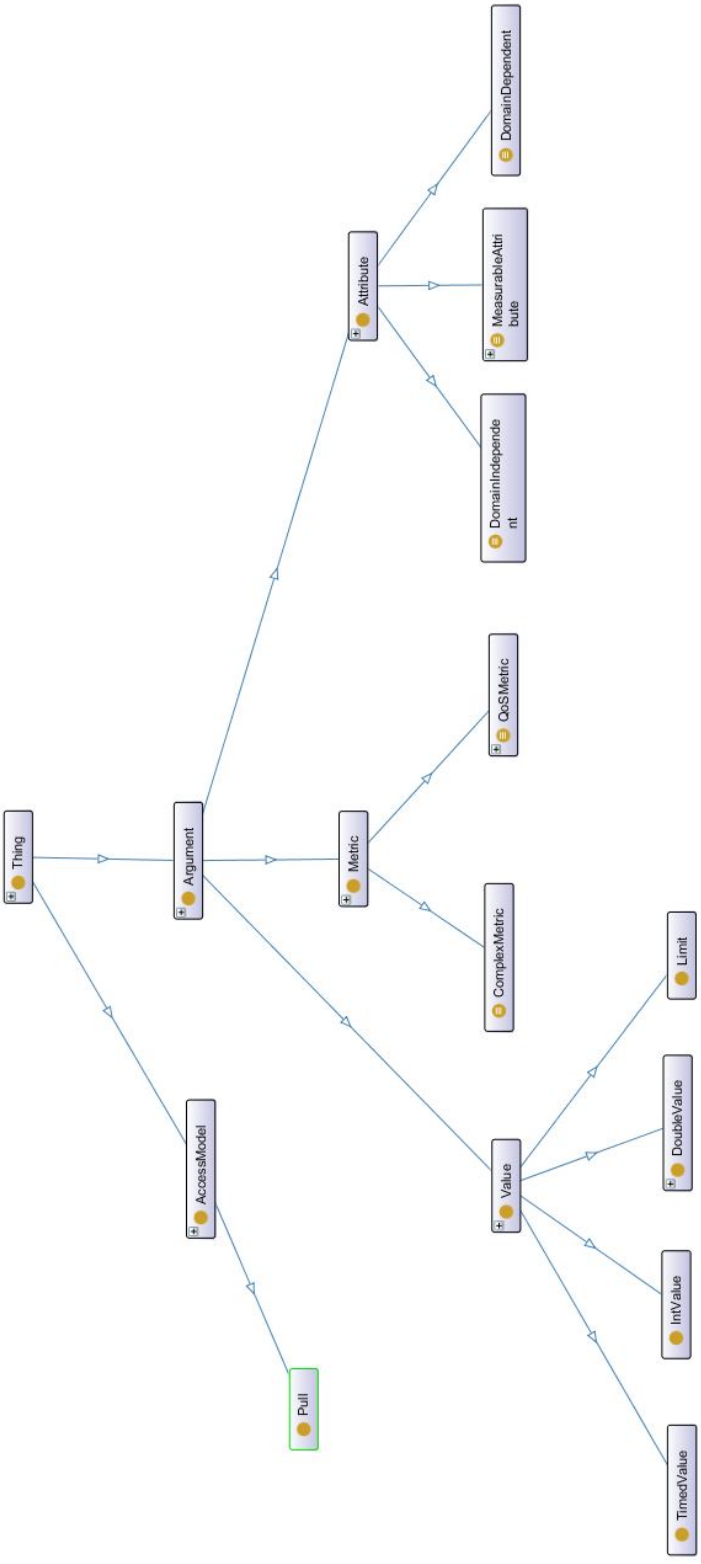


Figure 5.6: OWL-Q ontology schema for quality metrics

Chapter 6

Quality Metrics Aggregation

Each of the three Metrics Aggregators are the central components of each of the Monitoring Managers, providing the core functionality for the collection and manipulation of the monitored data. Also, in this section we describe the functionality and usage of a fifth aggregator that refers to Scalability, Elasticity, Adaptability and PaaS quality metrics. In next sections we describe the implementation of the aggregators, accordingly for each of the layers.

6.1 Service Layer Aggregator

As has been stated in Section 5.2 and shown in Figure 5.2 the aggregation part refers on step number five where the appropriate API clients make their calls in order to retrieve and store metric data. In Figure 6.1, we can distinguish the action of the service layer aggregation in four steps. Before continuing with the description of each of the steps we want to state that we have used KairosDB [52] as the ManagementDB instance. KairosDB uses Cassandra [53] to store its data. The main reason of selecting KairosDB is that we are capable of inserting, deleting and querying metrics and datapoints by actions supported by the KairosDB API client implementation which is compatible with Java language, in our case.

To continue with the description of the procedure, on the first step Prometheus API client calls are executed retrieving raw metric data every one minute. These calls are performed for the time period between the last five minutes. On the second step of the

Figure 6.1: Architecture of Service layer aggregator

procedure the actual aggregation is done indicating two calculation procedures:

1. Calculation of raw metric data, as prometheus provides metric data that are already aggregated by a non-applicable way based on our SM definition. Summarized aggregations are returned from Prometheus, thus in order to transform the data into raw metric values (as defined in SM), we are subtracting each time the previous summarized value from the following of it in order to obtain the according raw value along with each timestamp. This procedure is done as many times as the number of sum entries of the API client response for a specific raw metric that has been defined. For performance reasons once the number of data entries in the directory storage of Prometheus,Figure 5.2, reaches an upper limit we safely clean up the directory, as we have already parsed and pushed the raw values up to KairosDB. By following this procedure we minimize the number of sum entries that are going to be used for the calculation of the metric raw data.

2. Once the raw values are calculated we aggregate them based on the definitions of composite metrics made in SM.

Step three is not mandatory as we can skip it for the reason that we are retrieving monitoring data based on the time interval of the last five minutes. Although, we do follow step three in order to calculate the value of some metrics that need to have access on history data. More precisely, we need historic data for the calculation of MTBF and MTTF metrics that already have been pushed on KairosDB, thus on step three we are retrieving the appropriate historic metric data and then continue with the calculations for the values of MTBF and MTTF. In the end, during the fourth step we are actually pushing the values of raw and composite metrics in KairosDB. As shown in Figure 6.1, on steps three and four we have used KairosDB client API calls in order to retrieve and store data upon the instance of KairosDB. KairosDB also offers, the ability to aggregate values by defining certain aggregation variables. Although, we have decided to rely on aggregations actions that Prometheus monitoring tool offers, by the execution of Prometheus client API calls, and posses the instance of KairosDB only for storing metric data. Following this direction, we are not based on the implementation of the respective TSDB for aggregation actions, thus making the aggregation implementation undependable from the instance type of TSDB. On the technical part we have implemented seven Java classes in order to aggregate and store the metric data. We packed these classes as a jar file and run it as daemon which pulls data metrics every one minute as already has been stated.

6.2 Infrastructure Layer Aggregator

In the case of the Infrastructure layer aggregator we have followed the same implementation as the one described in Section 6.1 of Service Layer Aggregator implementation.

We have followed three steps in order to aggregate and insert data metric values in KairosDB, as Figure 6.2 shows. In the first step, by the usage of a JSON Exporter that

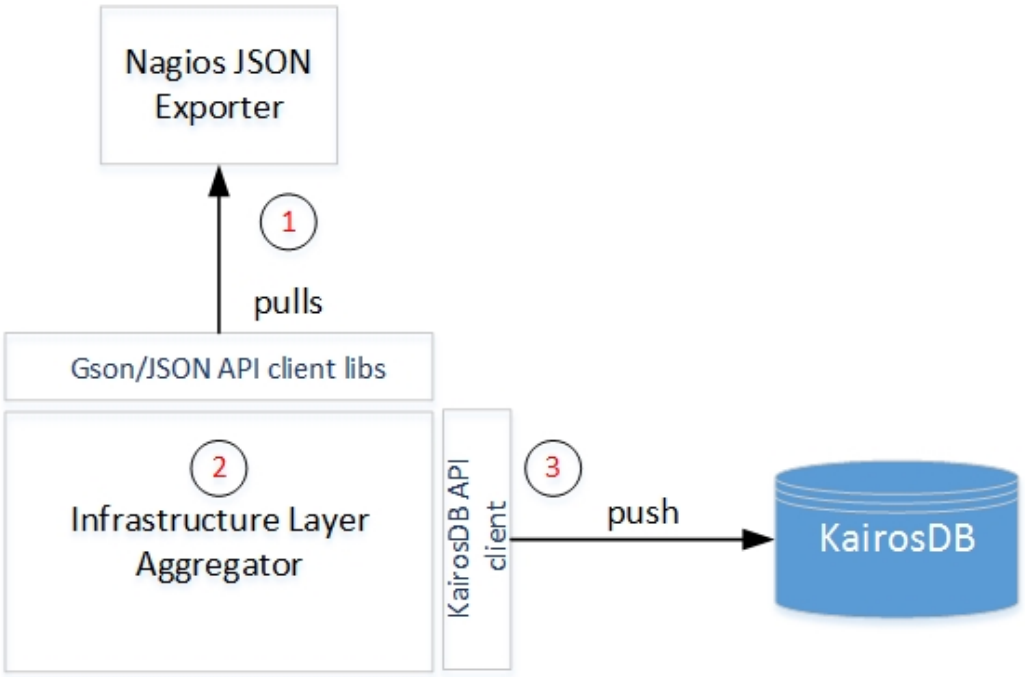


Figure 6.2: Architecture of Infrastructure layer aggregator

Nagios supports [54] and the usage of the appropriate Java client libraries that manipulate JSON objects (Gson [55]) we were able to retrieve data for each of the raw metrics. Frequency of pulling for the raw values of metrics happens every 60 seconds. At the second step the aggregation of the composite metrics happens. The reason for the selection of the one minute interval in order to aggregate and push metrics is, to be uniform with the frequency selection of service layer aggregator which also aggregates and push values every one minute. Aggregations are being done outside KairosDB, in plain Java classes, as the possible usage of a KairosDB aggregator would add unnecessary complexity to the implementation of the aggregator. Furthermore, we selected not to use KairosDB for the aggregations for the same reason as referred to the section of Service Layer Implementation. In the third step, once the aggregations in order to compute the composite metrics are done, we push the composite metric data in to the instance of KairosDB with the use of KairosDB client libraries.

6.3 Scalability/Elasticity/Adaptability and PaaS Aggregator

In the function layers of Service and Infrastructure we have defined quality attributes and quality metrics that are related with *Scalability*, *Elasticity*, *Adaptability* and PaaS (SEAP) aspects. We implemented the calculation formulas and the retrieval procedures of metrics related to the aforementioned quality attributes in a separate metric aggregator cause of the close relations that quality metrics have to each other. These relations are also stated

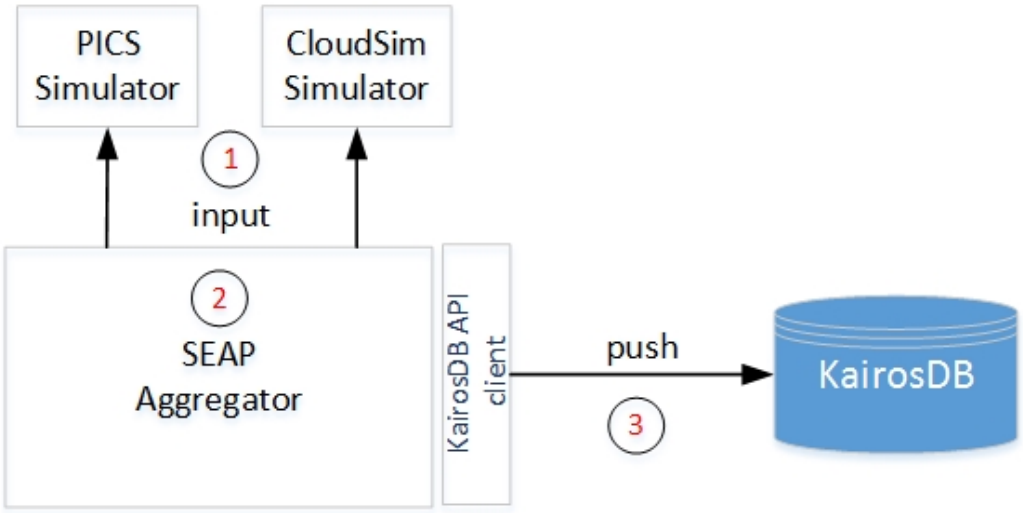


Figure 6.3: Architecture of SEAP aggregator

in the cross-layer dependency quality model where cross-layer dependencies between IM and SM exist.

Architecturally, we refer to SEAP aggregator as a sub-aggregator cause it implements calculation formulas that are stated in SM and IM quality layers and placed accordingly to the user VMs that SM and IM aggregators rely (Figure 4.2). As Figure 6.3 shows, SEAP aggregator follows three steps in order to retrieve, aggregate and push metrics in the according TSDB of the user VM. In the first step of the SEAP aggregation procedure, the aggregation component takes as input SEAP variables that are taken from PICS [56] and CloudSim [57] simulators. These simulators generate SEAP metric values that can be used for the calculation of the aggregated metrics. The reason that we choose to use cloud simulator tools in order to generate SEAP metric data is that we want to follow a generalized solution that let us to enable specific modeling of the Infrastructure (related to the private infrastructure that the actual services run). Based on the above we are able to investigate aspects such like scalability and provision of VMs without actually using any of the public cloud providers (EC2 [58] , Google Cloud [59] etc). Since the aggregation has finished, by executing the calculations of the formulas defined in the according metric quality models, the generated values are going to be pushed in the local and central ManagementDBs (see Figure 4.1). In Chapter 8, we are inferred on the potential of using real public cloud providers so as to get metrics from real time functioning infrastructures, in order to calculate scalability and provisioning metrics by the usage of service APIs that public cloud providers offer e.g Amazon CloudWatch [60].

In Table 6.1 there is detailed information of the raw SEAP variables being used, from which simulator they are taken from and which aggregation metrics are going to be calculated based on the incoming simulator metric values (raw metric *Scaling Started/Ended Dates* refer both on Scaling In and Scaling Out actions). A possible affect of the results of the SEAP metrics, could refer to the metric of NSLOV which is based on the SLOs

Raw SEAP metrics	**Composite SEAP metrics**	**Cloud Simulator**	
		PICS	CloudSim
Scaling point of interest	Scaling, Range,Utilization,Speed	✓	
Adaptations/ Scaling Actions	Scaling Precision, PrecisionOfAdaptation	✓	
Total Adaptations/ Scaling Actions	Scaling Precision, PrecisionOfAdaptation	✓	
Scaling Started Date	Reaction Time, MTTR, Adaptation Time, PerfScaleFactor, MTQR	✓	
Scaling Ended Date	Reaction Time, MTTR, Adaptation Time, PerfScaleFactor, MTQR	✓	
Scaling out Start Date	Avg,Precision,Elasticity ScalingOut	✓	
Scaling out End Date	Avg,Precision,Elasticity ScalingOut	✓	
Scaling in Start Date	Avg,Precision,Elasticity ScalingIn	✓	
Scaling in End Date	Avg,Precision,Elasticity ScalingIn	✓	
Avg Underprovisioned	Precision/Elasticity ScalingIn	✓	
Avg Overprovisioned	Precision/Elasticity ScalingOut	✓	
Successability Adaptation			✓
Deployment Time			✓
Re-Deployment Time			✓
Replication/ Allocation time VM-to-VM			✓
Migration time VM/soft.compont			✓

Table 6.1: Matching of SEAP metrics with Cloud Simulator

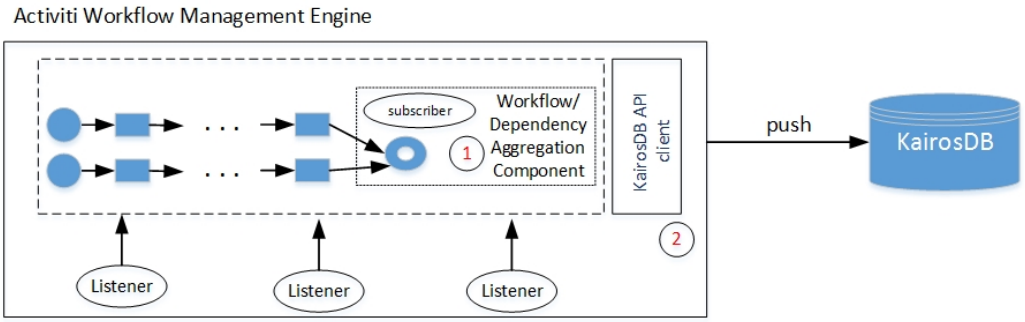


Figure 6.4: Architecture of Workflow aggregator

that have been posed for IaaS/SaaS, Figure 4.1, e.g, the max throughput of service X has to be greater than 10 req/sec after the scaling time point Z. If this requirement is fulfilled after the scaling/elasticticy action then the according SLO is valid, if not the number of NSLOV is increased by one.

Let us note here that the scaling and elasticity procedures are based on a configuration that matches the actual private and public cloud resources that we have used, Section 7.1.2, in order to be aligned and valid with the according configuration of the cloud simulators producing the raw metric values.

6.4 Workflow Layer Aggregator

There is a separation of the workflow aggregator and the dependency cloud aggregator. Both of them relies on the same VM but in the case of the workflow aggregator we aggregate values in order to compute metrics defined without the need of cross-layer dependency information.

In this implementation we have separated the aggregation and pushing of metrics in two steps. On the first step of the implementation we collect each of the raw metrics defined for the workflow layer and then aggregate them. This procedure is done at runtime once each of the workflows reaches their endpoint. When the flow reaches the endpoint, then the according listener is responsible for the collection and aggregation of the metrics. The reason that both of the workflows have the same endpoint, in Figure 6.4, is that the collection and aggregation part is the same for all the workflows operating within the wokrlfow management engine. On the second step once the aggregations are finished then, we used the KairosDB client API [61] which makes the appropriate calls on the KairosDB instance of the workflow layer VM.

We have decided to establish the gathering of the according metric information on the endpoint of each of the workflows as we care for workflows that have ended successfully, based also on the definitions in WM, regardless the fact of the failure of some of the tasks. In case of failures we follow three directions (a) we retrieve the raw metric data till the point of task failure from H2 Database that Activiti persists, (b) we instantiate a subscriber in order to retrieve the according dependency metric values from the layer of SM and (c)

Type of Aggregator / Quality Dimension	**Workflow Layer Aggregator**	**Dependency Aggregator**
Time Quality Dimension	✓	✓
Cost Quality Dimension	✓	
Reliability Quality Dimension		✓
Security Quality Dimension		✓

Table 6.2: Aggregators of Quality Dimension metrics

we push the according aggregated and raw metric values into KairosDB.

Metrics of the quality dimensions of reliability and security can be successfully computed by usage of the dependency metrics aggregator, due to the cross-layer dependencies defined in Dependency model between WM and SM. Table 6.2 indicates the responsible aggregators, that calculate and push the according composite and raw metric values, for each of the workflow dimensions defined in WM. Let us note here that the the values of cost for each of the workflow tasks are different for each of worklfow instances as they are being randomly selected between certain numbers (between 0 and 20000 units).

6.5 Dependency Cloud Aggregator

As discussed in the Section 3.4 there are two types of cross-layer dependencies that we take into consideration. In this implementation of the cross-layer dependency aggregator we have implemented the type of dependency where a metric in layer X can be used in order to calculate a metric in Y. More precisely metrics included are from SM and WM layers.

In order to pass the dependency metrics from lower to higher layers, we have used the implementation of the publish/subscribe system (see Section 5.5). We have selected as publishers the number of the VMs that were established with SM/IM related metrics and as a subscriber the VM that the Workflow Engine VM is functioning. The publishers placed in the user VMs are sending their data every one minute and the subscriber from the other hand retrieves these values whenever the workflow has reached it's ending point and then it passes these values to the dependency aggregator. We have selected to put the dependency aggregator and the subscriber in the end of the workflow,(see Figure 6.4) for the same reasons as explained in the Section 6.4.

As SM generates a number of raw and composite metric values that might include more than one data points, we are only interested in the retrieval of a group of metrics that concern the workflow instance that has just been executed. Thus, the moment the workflow has started, it immediately sends (a) the starting point *timestamp* to the user VM that SM is implemented and (b) the *id* of the workflow instance. Following this direction, publisher will only pass the metrics dependency values occurred between a certain period of time which is the starting point received from WM till the time where

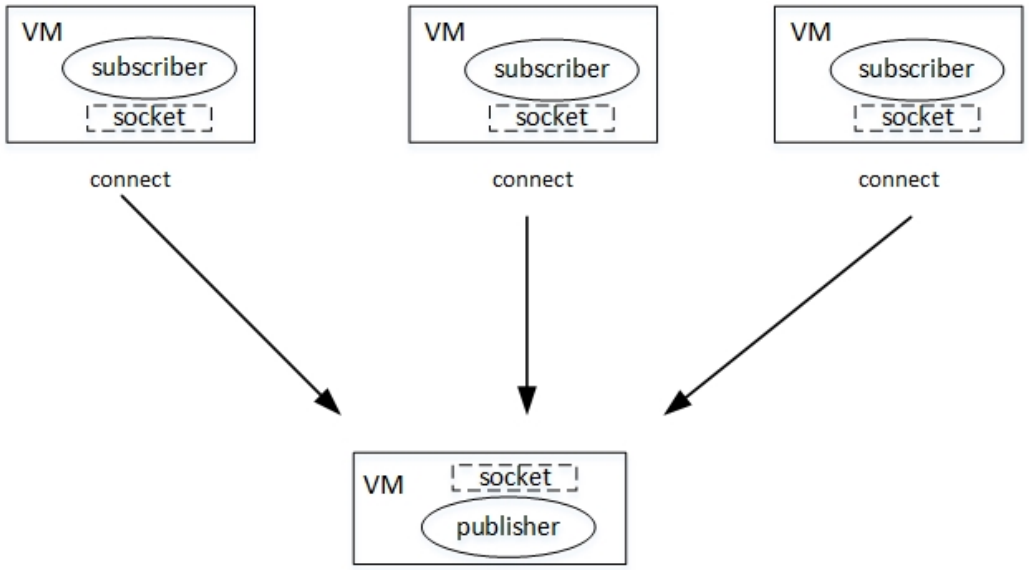


Figure 6.5: Publish/Subscribe implementation that transfers data from WM to SM layer

the workflow has ended. In order to send the starting timestamp point of the workflow and the workflow instance's id we have implemented a publish/subscribe system where the publisher is the WM and the subscribers are each of the userVMs, Figure 6.5. User VMs connect to a single publisher socket indicating a *one-to-many* relationship, in contrast to the implementation explained in Section 5.5 where *one-to-one* relationship between publisher and subscriber sockets is sustained.

Each time a cross layer dependency value for the calculation of dependency metrics is needed, we create an object instance of the Subscriber java class. Calling method of the subscriber belongs to the implementation of the dependency aggregator. For example, in order to retrieve the TaskDelayTimes metrics, defined in Time Quality Dimension, we make the according calls to connect to the publisher socket and then receive the aggregated Start and End Entity Delay Times metrics from SM, for each of the services and, for the time period where the workflow has started till the end of it. Listing 6.1 shows the creation of a subscriber instance and the according calls in order to retrieve and use the cross layer values for the calculation of the according dependency metrics.

```
1 Subscriber sub = new Subscriber(); /* subscriber creation */
2 sub.getTaskDelayMetrics(); /* retrieval of the cross-layer dependency value */
```

Listing 6.1: Subscriber object creation

Once the dependency values of composite cross layer dependency metrics are calculated and retrieved, then they are pushed in the local and central ManagementDBs.

Chapter 7

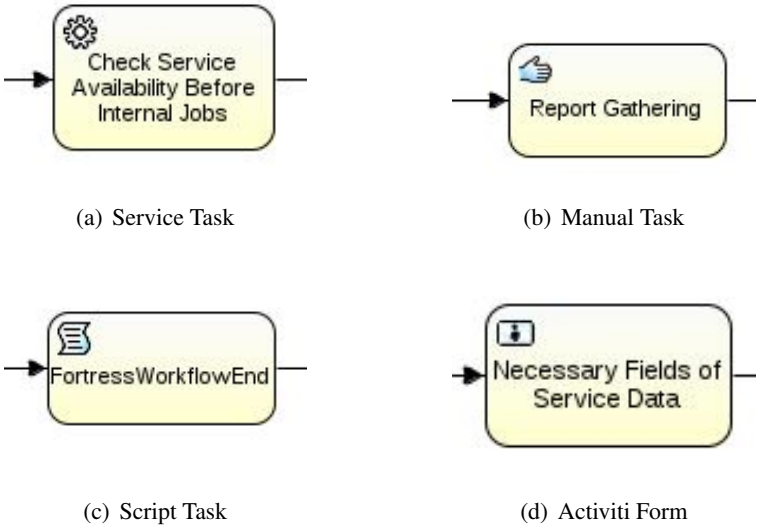
Evaluation and Validation

In this chapter we demonstrate (1) a mapping to actual external and internal services with regard to the workflow tasks as well as a mapping of internal services and software components to VMs at the Infrastructure layer. We expose the quality metrics calculations approaches by a running example which constitutes the procedures for the calculation of metrics for the three layers, and the cross-layer dependency metrics explicating how they are being computed and transferred from lower to higher levels, IM to SM and SM to WM. We (2) run experiments in order to measure the *execution response times* of the local/central managementDBs and of the publish/subscribe system for different amounts of workload in order to represent the performance of the proposed distributed monitoring system. Finally, (3) in our last experiment we measure the accuracy of the monitoring system.

7.1 Running Example

7.1.1 Workflow process description and Service mapping

Figure 7.2 shows the Activiti workflow process that we have executed. Its structure is identical to the one that we have shown in Section 3.1.1.1 of WM, for the calculation of *execution* and *delay* time of tasks. This workflow represents the use of a set of web-services that are part of standard-based access management system that provides role-based access control, delegated administration and password policy services with the ApacheDS [40] LDAP. Purpose of the execution of this workflow is to test the perviousness of each of the Service Pages which manage the actions of a different set of user groups. These actions include create, read, update and delete (CRUD) actions upon the data provided by the according service. A general rule that applies to our experimental performance and validation is that for each separate workflow there is a set of services that runs on a unique server e.g, fortress-web running on a certain instance of tomcat server. Following this direction we have established a mapping between workflows and services. A group of workflow instances of the same workflow indicate a set of services, although a group of workflow instances of different workflows indicate sets of different services.



(a) Service Task (b) Manual Task

(c) Script Task (d) Activiti Form

Figure 7.1: Workflow Activiti entities

Figure 7.1 demonstrates the four types of Activiti entities that are being used, more precisely (a) *service tasks* which are internally connected to java classes that corresponds to the functionality of the according internal services (b) *manual tasks* define tasks that are external to the BPM engine and that the model work that is done by somebody, which the engine does not need to know of, nor is there a system or UI interface (c) *script tasks* are automatic activities which are executed when a process execution arrives and (d) *Form* types are forms that can be so that the user is able to fill with values mandatory variables for the workflow process.

The workflow process that we have executed include eleven services/tasks. In Figure 7.3 we demonstrate the eleven page links that were used as internal services mapping to each of the workflow tasks. Table 7.1 represents the mapping between workflow tasks and service links.

Besides the eleven services tasks that correspond to each of the eleven services there are also (a) a service task that checks if the name of the service that was given as input is included in the list with the available web services , if not the workflow ends (b) a form in which the necessary fields are filled , like the IP of the host of services, the port of it and the number of the users that are going to access the services (c) a service task that checks if the values of IP and port produce a successive connection, if it is not able to perform a successful connection then the workflow ends (d) a service task that executes some failed and some successful log-ins into fortress-web (e) a manual task indicating the manually action of gathering reports of actions for each of the page/links and finally (f) a script task which grants the signal that the workflow will end in the next step.

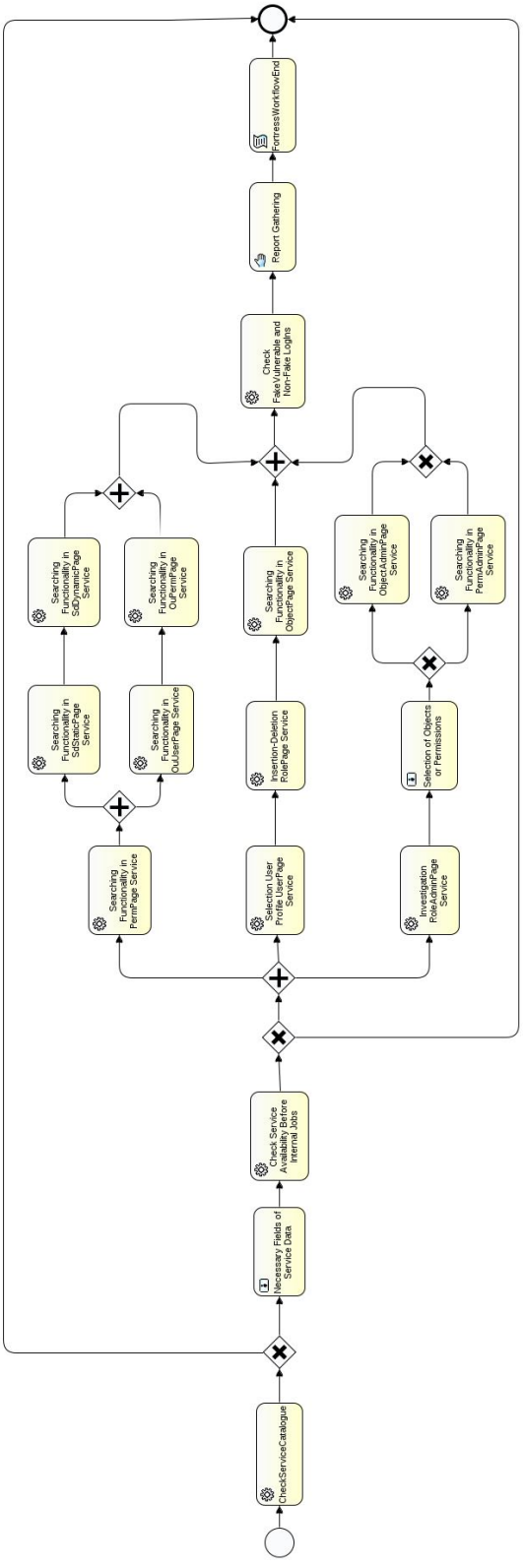


Figure 7.2: Running example of Activiti Workflow process

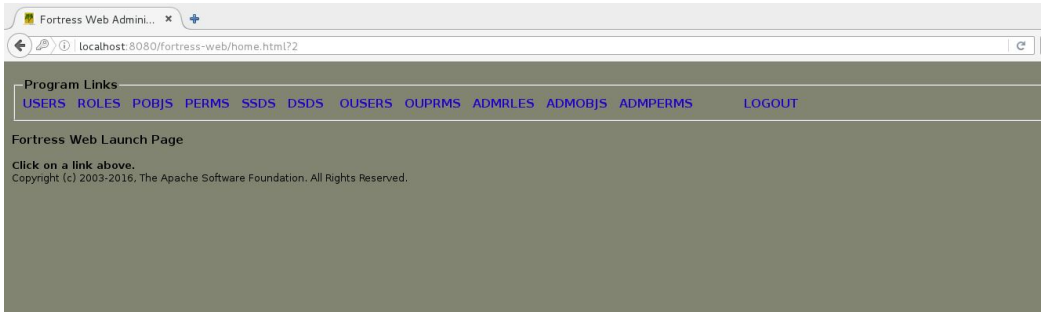


Figure 7.3: Fortress Web Services

Service Tasks	**Internal Service Pages/Links**
Selection User Profile UserPage Service	USERS
Insertion-Deletion RolePage Service	ROLES
Searching Functionality in ObjectPage Service	POBJS
Searching Functionality in PerPage Service	PERMS
Searching Functionality in SdStaticPage Service	SSDS
Searching Functionality in SdDynamicPage Service	DSDS
Searching Functionality in OuUserPage Service	OUSERS
Searching Functionality in OuPermPage Service	OUPRMS
Investigation RoleAdminPage Service	ADMRLES
Searching Functionality in ObjectAdminPage	ADMOBJS
Searching Functionality in PermAdminPage Service	ADMPERMS

Table 7.1: Mapping between Service Tasks and internal service pages/links

7.1.2 Running Environment

Workflow instances of the running examples were carried out in two cloud infrastructures. The first one is a private cloud infrastructure offered by the PaaS of VMWare in which we have provisioned in total four VMs hosted by a PC of Intel Core i7-4 Cores 2.50 GHz, 16 GB Ram, 300 GB SSD Hard Disk, running Windows 10 64-bit Operating System (OS). Two VMs were provisioned with characteristics of 2 vCPUs, 2 GB Ram, 15 GB hard disk and OS of Centos7 64-bit. The remaining two VMs had the characteristics of 2 vCPUs, 4 GB RAM and 20 GB hard disk and OS of Centos7 64-bit. In the public cloud infrastructure of University of Crete Data Center(UCDC) we have used one VM of the middle category indicating the properties of 2 vCPUs, 4 GB RAM, 100 GB Hard Disk drive and OS of ubuntu 14.04 64 bit.

7.1.3 Standard Workflow Running Example

In this section we are referring to the first experiment depicting the running of the workflow process in which we had adapted a standard workload of 20 requests/minute and one user. The workload for each of the tasks/web services shown in Table 7.1 was dispatched with the usage of Selenium [62] which is a portable software testing framework for web services and applications. The execution of the workflow process in order to complete lasted 10 minutes.

Algorithms responsible for the calculation of task/workflow execution time, taks/-workflow delay and task fidelity are being executed at the end point of the worklfow. As Workflow and Task Fidelity are based on history values and user constrains placed by the user, we had provided such constraints based on metrics of *failureTime*, *availability* and *execution* separately for tasks and workflows. Thus, once the current values of the metrics are available at the end point we execute the appropriate methods and store the values task/workflow fidelity for the current run to the local and central TSBDs instances. We have executed the according calculation formulas for the workflow authentication, task/-workflow availability and task/workflow delay times presented in WM once the pub/sub system passed the values from the lower level of service to the higher level of workflow as Figure 7.4 explicates.

Tables A.3, A.2 and A.1 represent the average metric data values of the metrics that are calculated for the current run of the workflow. Some metrics are annotated with the letter h; this means that the calculation of those is based on historic values, so as to indicate the comportment that this workflow has, till the last execution of it. We assume that actions in order to produce SEAP values have been conducted prior to the start of the workflow and is not part of it. Our *main goal* is to show that dependency values can be retrieved and passed from the lower levels, to higher levels as has been done between SM-WM. Thus, when the workflow process of fortress-web starts, cross layer dependency metric data between IM and SM (Section 6.3) have already been stored in the according TSDB.

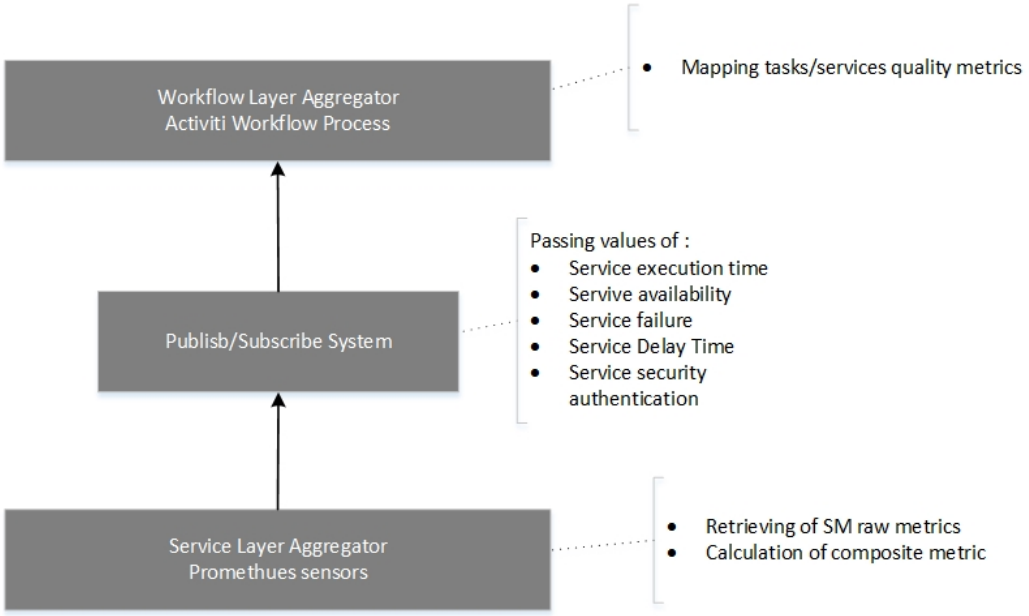


Figure 7.4: Passing values from SM up to WM layer

7.2 Performance Experiments

We have experimentally measure some basic metrics related to the web services but also on metrics related to the distributed monitoring system. Purposes of the experiments are (a) to validate the results of the monitoring system by explicating the performance quality metric result values (b) to measure the performance of the distributed monitoring system by the calculation of specific metrics and (c) to calculate the percentage of accuracy that the monitoring solution offers. All of the experiments were conducted based on the running environment described in Section 7.1.2. Let us also note that an experiment can be constituted by more than one runs.

7.2.1 Selenium workload environment

Most of the experimenters performed by one to four concurrent users, upon accessing the web services. We have supported this range of workload by the adoption of a Selenium Grid. We have instantiated a selenium grid of four VMs, as Figure 7.5 depicts. The workload is proportionated to the number of the simultaneously web browsers, from each of the separate VMs, accessing the web services. Thus, it's time an experiment is performed with the addition of one more user, we used one of the available selenium clients, instantiating a web-browser. In order for the client to work properly first, it has to connect to the selenium hub which was deployed on the VM that was the host of the WF engine. After a successful connection with the hub the workload is directed upon VM 1 client where the web services are being deployed and run.

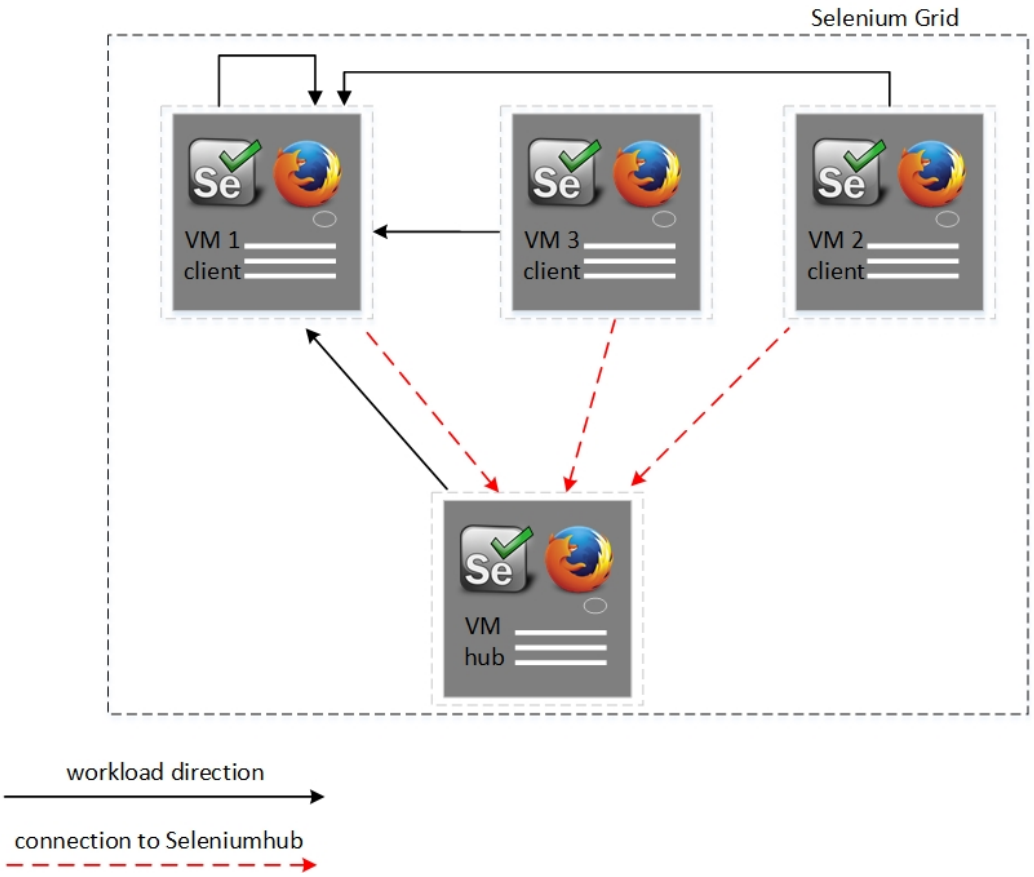


Figure 7.5: Connection between Selenium clients and Selenium hub

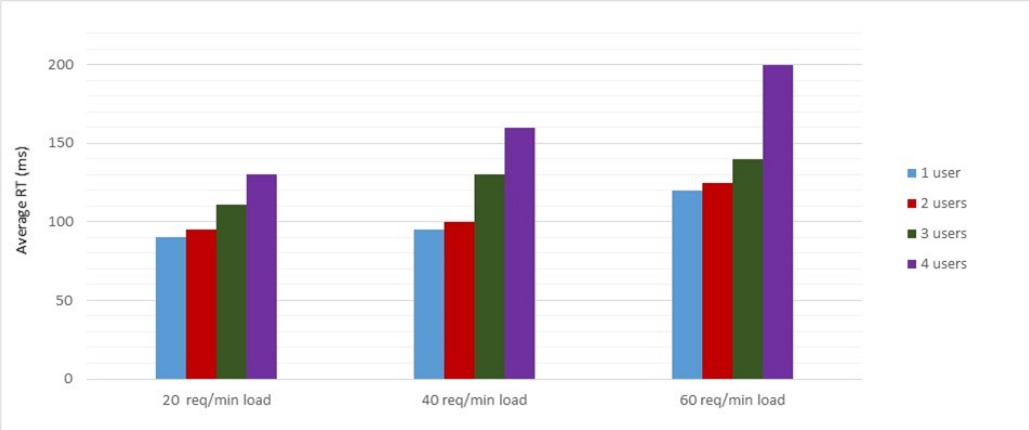


Figure 7.6: Average Response Time between 1,2,3 and 4 simultaneous users

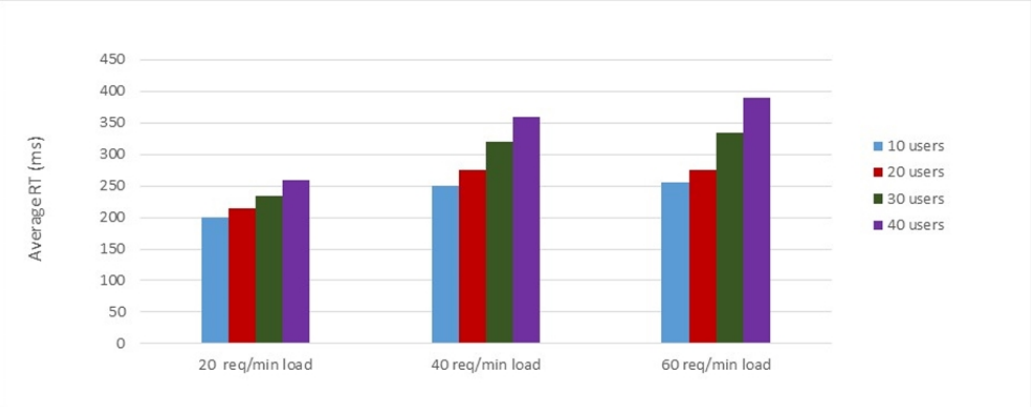


Figure 7.7: Average Response Time between 10,20,30 and 40 simultaneous users

7.2.2 Experiments

All of the experimenters are based on the workflow process presented in Section 7.1 .

7.2.2.1 Results Validation

In the first experiment we have performed the measuring of the *average response times* of the set of services of fortress-web with different amounts of workloads for each of the users accessing the services simultaneously. First, we ran the workflow with the workload of 20 req/min for each set of simultaneous users and then we follow the same pattern for 40 req/min and 60 req/min workloads.

As expected, the response time is increasing according to the number of simultaneous users and the number of workload requests/sec. Figure 7.6 indicates that there is an increasing difference between the number of simultaneous users. In the workload of 40 req/min the difference between two and four simultaneous users is bigger than 60 mil-

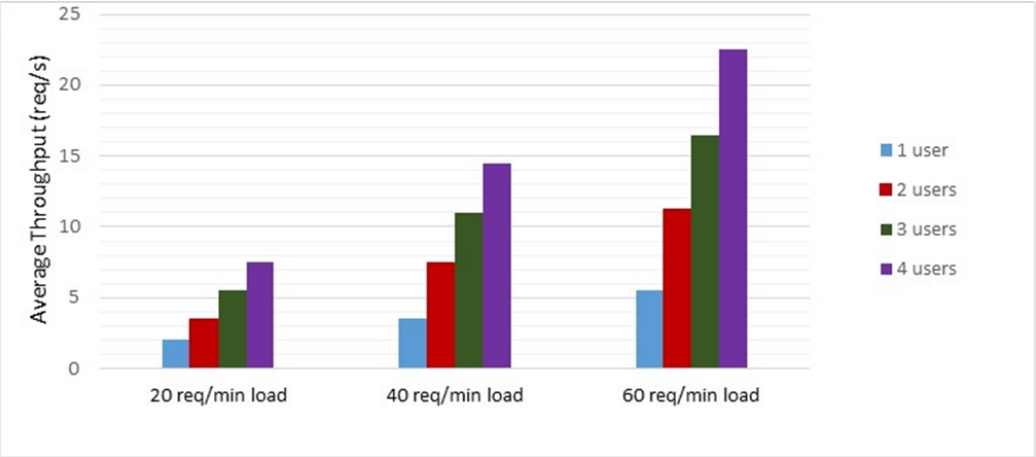


Figure 7.8: Average Throughput between 1,2,3 and 4 simultaneous users

liseconds and for the workload of 60 req/min is even bigger than 70 ms. This indicates that the of response time between the different numbers of simultaneous users is bigger than the amount of workload that the services can tolerate. The reason behind this could be the number of simultaneous threads that Java instantiates in order to serve the requests, thus having a late response time.

Our next experiment has been executed under the same conditions as the first experiment. Purpose of this experiment is to validate the throughput of services. Average throughput time has been calculated by the type of $\frac{NumberOfWorkload*NumberOfUsers*NumberOfServices}{60(sec)*NumberOfRuns}$. For this experiment we have done two runs for each number of simultaneous users along with their according workload. Reason of multiplying by 11 is that we have 11 service components that the users where requesting in each run (see Table 7.1). In Figure 7.8 we can observe that the throughput values change, the same way as the monitoring procedure depicted in the first experiments, indicating that the throughput difference between the number of simultaneous users is *proportional* to the difference between the workloads. The three types of workloads have an average difference of 8 milliseconds whilst the difference between the concurrent users could be up to 9 milliseconds. This is acceptable by the means that as the number of simultaneous users increases, so is and the number of the requests/s that the system is going to receive, according to each workload.

In Figures 7.7 and 7.9 we have validated the behaviour of the services when there is an increase in the number of users, concluding to proportional results (for the according workload), with one to four users. We have used ten times more users(10, 20, 30 and 40) for the experiments of response times and throughput and concluded that the response time latency and the throughput are also being analogously increased based on the detained workload.

The purpose of the next experiment is to investigate the relationship between the response time and the throughput of the set of web services of Fortress-Web for each of the four cases of different number of concurrent users, with workload of 60 req/min. Response time is the time needed to serve a client's request, while throughput is the number

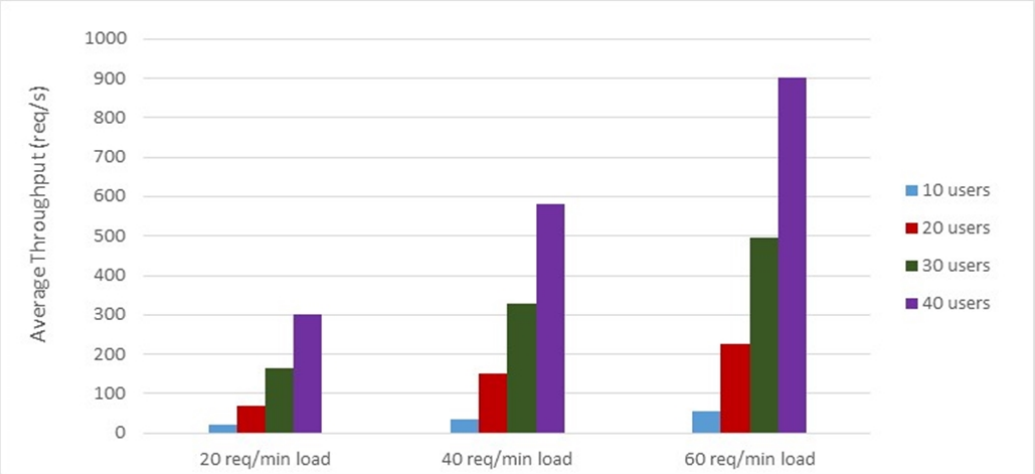


Figure 7.9: Average Throughput between 10,20,30 and 40 simultaneous users

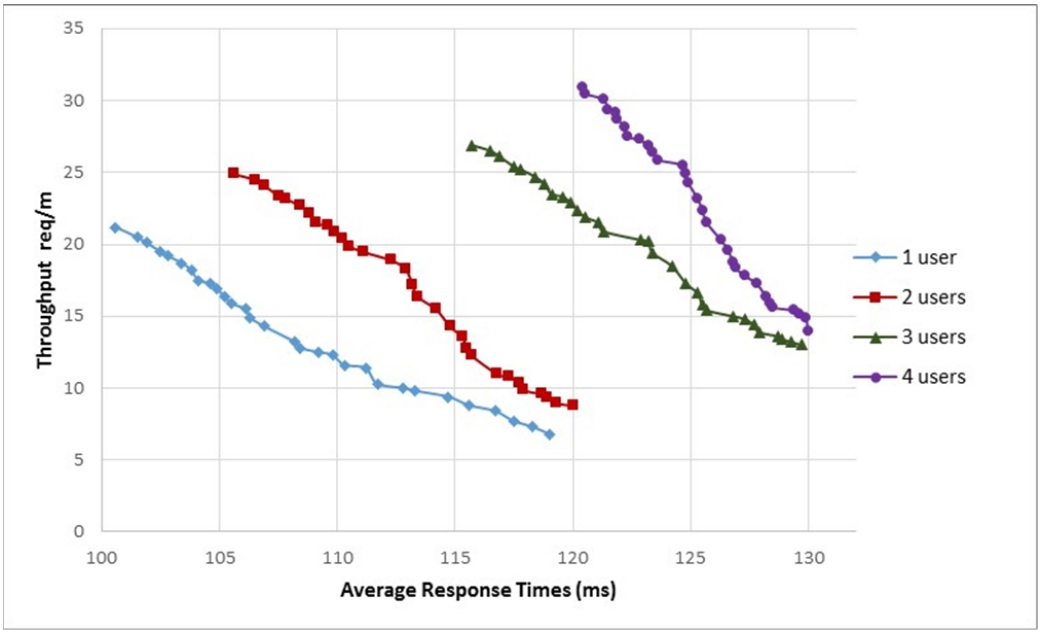


Figure 7.10: Relationship between response time and throughput

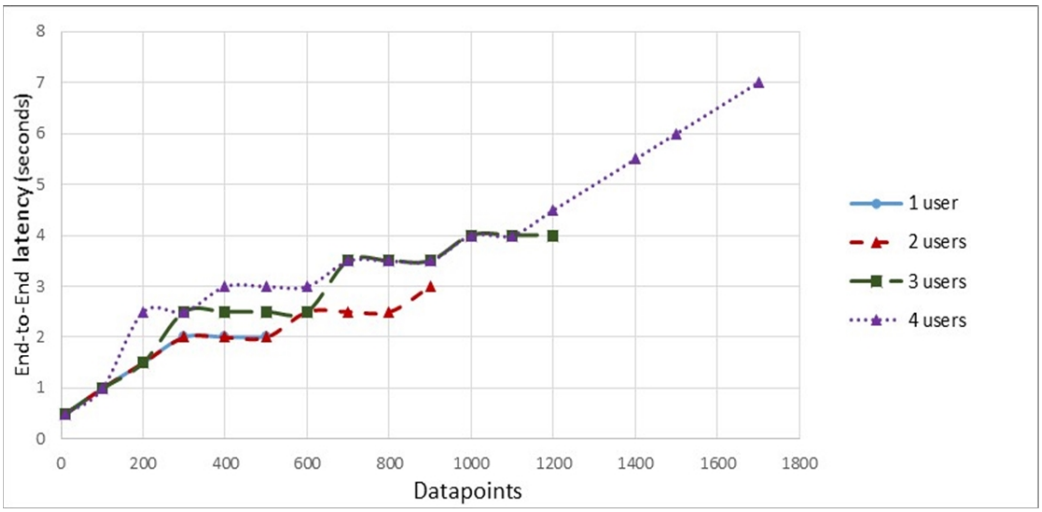


Figure 7.11: Monitoring system's end-to-end latency

of requests that can be served in unit of time. What we actually expect is these two metrics to be reversely proportional, Figure 7.10. We executed a separate run for each of the number of simultaneous users and observed that each run validate exactly the relation between the two metrics, analogously of course with the number of the concurrent users. Thus, as the response time gets bigger the throughput get lesser.

7.2.2.2 Monitoring system performance

The next three experiments that have been conducted evaluate the performance and the accuracy of the distributed monitoring system.

On our first experiment for this section, we have measured the end-to-end latency, stated as the total amount of time an HTTP query request takes from receiving to sending back the response, in relation to the datapoints that have been queried. The end points refer to the private userVMs, that Service, Infrastructure and Workflow monitoring managers are functioning, and the central KairosDB deployed in the public cloud infrastructure. Datapoints are defined as the numbers of values that have been recorded for the calculated metrics. We have followed the same procedure of runs as with the previous experiments for the workload of 60 req/min. In Figure 7.11 we indicate the fact that as the number of datapoints increases so is the latency. We can observe that the main latency time for the $\frac{2}{3}$ of the datapoints has been observed between 0.5 and 4 seconds and only in the extreme case of the 4 simultaneous users the query latency is up to 7 seconds, which is still acceptable given the fact of the increased number of datapoints up to 1650. Thus, as the number of simultaneous users increases, so is the number of cross-layer dependency data metric values, thus there is also a slightly increase in the delay of pub/sub system, due to number of connections that are being done between publisher and subscribers.

For the next experiment, Figure 7.12, we have measured the latency of the publish/-subscribe system implementation in relation to the number of the datapoints being pub-

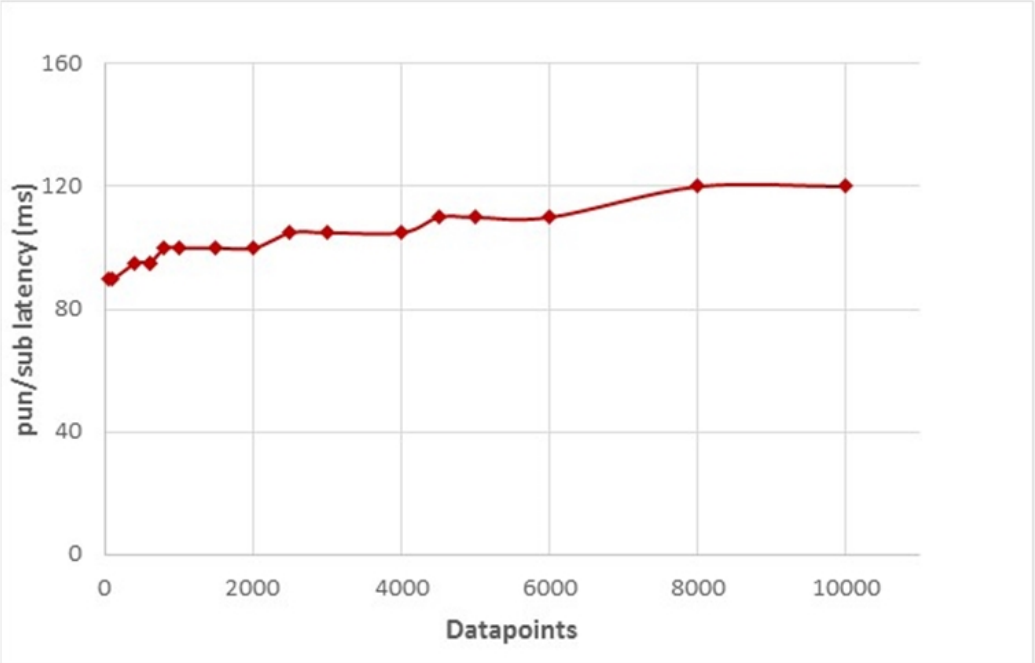


Figure 7.12: Publish/Subscribe system's latency

lished/subscribed. As the number of datapoints (being defined as in the previous experiment), are increased so is the latency of the pub/sub system. Although, the latency values are quite acceptable as the highest latency is spotted at 120 ms for 10000 where in 2000 datapoints we had a latency of 90 ms, thus making the difference of 30ms between 2000 and 10000 datapoints quite acceptable. The arousal of the datapoints has been based on the four main types of runs (with one, two, three and four concurrent users of 60 req/min) being run consecutively for this experiment, without cleaning up the monitoring data from previous runs, reaching the value of 10000 datapoints.

The last experiment evaluated the accuracy of the distributed monitoring system. More precisely we have executed seven runs, where each one those represented a workflow instance of the according run. In each of these runs we have calculated the accuracy of the monitoring system, produce by the division of the number of $\frac{Metrics+Datapoints(received)}{Metrics+Datapoints(expected)}$. These seven runs where made with a workload of 60 req/min and in each one them we have increased the number of user by one (starting with one user). As we can observe in Figure 7.13 the system does not fail under 90% of accuracy. The 10% percent loss of metrics and datapoints is due to the frequency of the monitoring system pulling data metrics values. We can tackle that by increasing the frequency of pulling values (according to Section 8 of Future Work, values of pulling frequencies for each of the quality metrics, are going to be derived by an updated OWL-Q ontology schema). In Table 7.2 we represent the *accumulated* values of (a) the expected metrics and datapoints, (b) the number of metrics and datapoints that actually derived and (c) also metrics that were not probed for each of the workflow runs. In overall, for these seven runs, we can observe

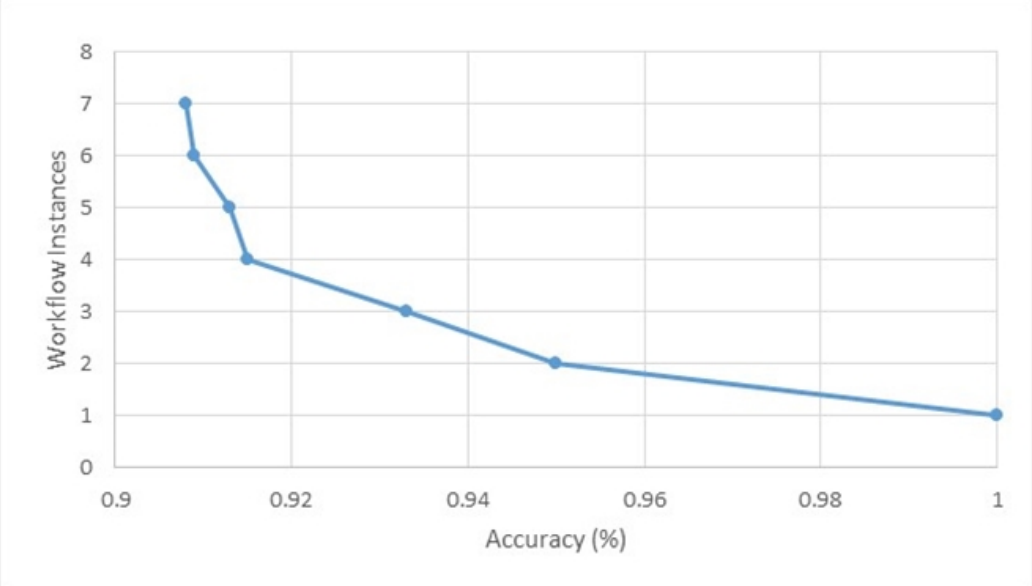


Figure 7.13: Accuracy of the Monitoring system

that the deviation of the lost metrics is approximately 2 and the deviation of the lost datapoints is approximately 40, which is quite acceptable as we made the experiment under the maximum workload of 60 req/min.

In a nutshell, we conclude on the fact that, the physical architecture that has been presented and the placement of the according software components upon the two cloud sectors have generated acceptable performance and accuracy aspects both for the web services and the distributed monitoring tool.

Workflow instances	Metrics/datapoints expected	Metrics/datapoints pulled	Metrics not received	Datapoints not received
1st	483	483	0	0
2nd	934	886	2	46
3rd	1326	1236	2	88
4th	1779	1628	4	147
5th	2122	1937	4	181
6th	2498	2270	5	223
7th	2844	2582	5	257

Table 7.2: Expected and received metrics and datapoints

Chapter 8

Conclusions and Future Work

In this thesis, we have demonstrated three metric quality models along with a fourth one indicating the cross-layer dependencies and relations between quality metrics of the three QMs of WM, SM and IM. QMs that have been proposed are being covered by defining quality terms describing each of the layer's metric composability. To address that, we have created computation formulas that can be used to assign values on cross-layer depended-metrics, defined in the according Dependency QM, that could not been calculated unless dependencies were not defined. Based on the QMs defined we have developed a distributed cross-layer monitoring system by the usage of two, third party monitoring tools of Prometheus for Service Layer, Nagios for the Infrastructure layer; we have also leveraged the monitoring capabilities of the Activiti workflow engine in order to retrieve monitoring data from the Workflow layer. Choice of Prometheus as a third party monitoring tool was based on an extensive comparison between various service monitoring tools that had to fulfill certain selection criteria in order to be chosen as the main pulling software for data metric.

We have implemented and use metric aggregators where each one of them correspond up to the QMs of Worklow, Service, Infrastructure and Dependency. Metric aggregators are the central components of the monitoring solution that we propose, as they are the ones that implement the metrics computation formulas that are being defined in each of the layers and are also, responsible for the pushing of metric data in the according ManagementDBs. Regarding the type of ManagementDB that we were about to use, we had the options of TSDBs and Event-based Databases. After the comparison that we have made we concluded that the selection of a TSDB type was the most appropriate choice for us, as we do not concern about run-time events but we only care for the pulling of run time raw data. Our comparison indicated that the usage of an event-based database is appropriate in occasions where KPIs and quality aspects concerning the Business layer are in place, which in this thesis was not included.

The physical architecture that has been proposed is based on a hybrid multi-cloud environment which is characterized by the modularity of the internal components, such as aggregators and the according monitoring sensors. We have validated the proposed monitoring system by giving a running example based on a workflow process ,with a complex

structure, which is compromised by a certain set of web services, with different number of workload and simultaneous users. We have accomplish the use concurrent users and push the great amount of workload on the web services, by the usuage of Selenium [62] which is a suite of tools specifically for the automation of web browsers. We leverage this attribute and created a Selenium Grid [63] consisted consisted of four VMs, in which we where able to instantiate a number of web browsers that had the role of web services users. In the end, we demonstrated that (a) the performance overhead of the developed monitoring tool is quite tolerated based on the physical architecture that is subject to and (b) from our experimental results, we conclude that the proposed types are precise enough and validate the implementation approach that we have followed.

In order to give an integrated general monitoring solution that will also include actions of adaptation based on SLAs, the process for (a) the integration with the distributed monitoring system framework [1] is on future plans (b) the intension to update our monitoring solution in order to be compatible with any service that need to be monitored, providing this way an adaptable monitoring procedure (c) the implementation of an event parser log that identifies possible vulnerabilities and security incidents in order to obtain real-life security metric data and (d) the extension of our OWL-Q schema in order to be able to fully describe the metrics domains and data model, that we are dealing with. Furthermore, we are going to enrich our QMs and our dependency model, so as to consider additional quality aspects mapping to the consideration of new domain-independent metrics and extend our ontology schema in order to facilitate an even more formally data model that will include e.g, the pulling frequency for each of the quality metrics. Also, we are going to consider additional metrics in order to give an extended validation of the approach according to specific use cases and highlight that the current structuring of the QMs is appropriate/suitable and also evaluate the distributed monitoring system with multiple hybrid clouds and even more workload and number of simultaneous users.

Finally, a major future objective is the creation of a *Business QM*, with regard to the the Business functional layer which could include quality terms in order to specify procedures that deals with the retrieval, processing, transformation and management of the services layer data. This will realise monitoring issues that will deal with business quality dimensions, attributes, metrics and KPIs. The purpose of defining a Business QM and provisioning a monitoring solution for it, is to support the notion of *Conceptual Analytics*. The common denominator is the requirement to transform the source (data, information, knowledge) into target format in order to support comprehensibility level of produced results for specific SMEs within the domain. *Conceptual Analytics* will (1) match the concepts from the domain ontology and the values from semantically enriched logs and (2) enable assimilation of the findings extracted from logs to the Business layer. In order to yield results from (1) and (2) and make them processable by humans we need to cover the subject of graphical representation of ontology concepts. This will require to (a) elaborate on appropriate expressive power of a reference ontology required to explicitly enable the transformation between modeling languages describing the source (logs) and target, (b) to address the domain/user specific notation for the resulting concepts and (c) to setup transformation framework on the meta level, thus enabling semi-automatically transformation of the corresponding models and attribute values by semantic annotation.

This will enable graphical representation of the execution results in the initial business process – e.g. in form of tachometers wrt. to compliance with SLAs.

Bibliography

[1] C. Zeginis, K. Kritikos, P. Garefalakis, K. Konsolaki, K. Magoutis, and D. Plexousakis, "Towards cross-layer monitoring of multi-cloud service-based applications," in *Service-Oriented and Cloud Computing - Second European Conference, ESOCC 2013, Málaga, Spain, September 11-13, 2013. Proceedings*, 2013, pp. 188–195.

[2] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benbernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro, "A survey on service quality description." *ACM Comput. Surv.*, vol. 46, no. 1, p. 1, 2013.

[3] K. Bratanis, D. Dranidis, and A. J. H. Simons, "Slas for cross-layer adaptation and monitoring of service-based applications: a case study." in *QASBA*, ser. ACM International Conference Proceeding Series, D. Bianculli, S. Guinea, A. Metzger, and A. Polini, Eds. ACM, 2011, pp. 28–32.

[4] "Broker@cloud, deliverable 20.1, sate of the art and reasearch baseline," 2013.

[5] D. D. K. Bratanis and A. J. H. Simons, "An extensible architecture for the run-time monitoring of conversational web services," ser. D. Karastoyanova, R. Kazhamiakin and A. Metzger (eds.), Proc 3rd. Int. Workshop on Monitoring,Adaptation and Beyond, ACM New York, 2010, 9-16, 2010.

[6] K. Alhamazani, R. Ranjan, P. P. Jayaraman, K. Mitra, C. Liu, F. A. Rabhi, D. Georgakopoulos, and L. Wang, "Cross-layer multi-cloud real-time application qos monitoring and benchmarking as-a-service framework." *CoRR*, vol. abs/1502.00206, 2015.

[7] J. M. A. Calero and J. Gutierrez, "Monpaas: An adaptive monitoring platformas a service for cloud computing infrastructures and services." *IEEE Trans. Services Computing*, vol. 8, no. 1, pp. 65–78, 2015.

[8] M. García-Valls, I. R. Lopez, and L. Fernández-Villar, "iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems." *IEEE Trans. Industrial Informatics*, vol. 9, no. 1, pp. 228–236, 2013.

[9] R. Kazhamiakin, M. Pistore, and A. Zengin, "Cross-layer adaptation and monitoring of service-based applications." ser. Lecture Notes in Computer Science, vol. 6275, 2009, pp. 325–334.

[10] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein, "Multi-layered monitoring and adaptation." in *ICSOC*, ser. Lecture Notes in Computer Science, vol. 7084. Springer, 2011, pp. 359–373.

[11] K. P. Joshi, A. Joshi, and Y. Yesha, "Managing the quality of virtualized services," in *2011 Annual SRII Global Conference*, March 2011, pp. 300–307.

[12] J. Cardoso, A. Sheth, and J. Miller, "Workflow Quality Of Service," LSDIS Lab, Computer Science, Universtity of Georgia, Athens GA, USA, LSDIS Lab, Computer Science, Universtity of Georgia, Athens GA, USA, Tech. Rep., March 2002.

[13] A. K. Bardsiri and S. M. Hashemi, "Qos metrics for cloud computing services evaluation," *International Journal of Intelligent Systems and Applications*, pp. 27–33, 2014.

[14] M. Becker, S. Lehrig, and S. Becker, "Systematically deriving quality metrics for cloud computing systems." in *ICPE*, L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó, Eds. ACM, 2015, pp. 169–174.

[15] N. R. Herbst, S. Kounev, and R. H. Reussner, "Elasticity in cloud computing: What it is, and what it is not." in *ICAC*, J. O. Kephart, C. Pu, and X. Zhu, Eds. USENIX Association, 2013, pp. 23–27.

[16] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, "Modeling quality of service for workflows and web service processes," Oct. 2002.

[17] J. A. Miller, D. Palaniswami, A. P. Sheth, K. Kochut, and H. Singh, "Webwork: Meteor2's web-based workflow management system." *J. Intell. Inf. Syst.*, vol. 10, no. 2, pp. 185–215, 1998.

[18] "Activiti workflow process engine," http://www.activiti.org, 2016.

[19] J. M. Gómez-Pérez, E. García-Cuesta, J. Zhao, A. Garrido, and J. E. Ruiz, "How reliable is your workflow: Monitoring decay in scholarly publications." in *SePublica*, ser. CEUR Workshop Proceedings, A. G. Castro, C. Lange, P. W. Lord, and R. Stevens, Eds., vol. 994. CEUR-WS.org, 2013, pp. 75–86.

[20] P. Cedillo, J. Gonzalez-Huerta, S. Abrahão, and E. Insfrán, "A monitoring infrastructure for the quality assessment of cloud services." in *ISD*, D. Vogel, X. Guo, X. Ju, H. Lan, C. Barry, M. Lang, H. Linger, and C. Schneider, Eds. Association for Information Systems, 2015.

[21] I. O. for Standardization (ISO), *ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)*, Std., 2005.

[22] "Key performance indicator (kpi)," https://en.wikipedia.org/wiki/Performance_indicator, 2016.

[23] K. Kritikos and D. Plexousakis, "Requirements for qos-based web service description and discovery." *IEEE Trans. Services Computing*, vol. 2, no. 4, pp. 320–337, 2009.

[24] "Navigation timing,w3c recommendation," https://www.w3.org/TR/navigation-timing/, 2012.

[25] P. Trimintzios, "Measurement frameworks and metrics for resilient networks and services: Technical report systems," European Network and Information Security Agency (ENISA), Tech. Rep. [Online]. Available: https://www.enisa.europa.eu/publications/metrics-tech-report/at_download/fullReport

[26] Wikipedia, "Hypervisor," 2016. [Online]. Available: https://en.wikipedia.org/wiki/Hypervisor#Classification

[27] "Time series database," https://en.wikipedia.org/wiki/Time_series_database, 2016.

[28] "Real user monitoring," https://en.wikipedia.org/wiki/Real_user_monitoring, 2016.

[29] "Datadog," https://www.datadoghq.com/, 2016.

[30] "New relic," https://newrelic.com/, 2016.

[31] "Prometheus monitoring tool," https://prometheus.io/, 2016.

[32] "Telegraph," https://www.influxdata.com/time-series-platform/telegraf/, 2016.

[33] "Statsd," https://github.com/etsy/statsd, 2016.

[34] "Bosum server," https://bosun.org/quickstart, 2016.

[35] A. Osmani, "Timing js," https://github.com/addyosmani/timing.js/, 2013.

[36] "Apache fortress," http://directory.apache.org/fortress/, 2016.

[37] "Apache foundation project," https://projects.apache.org/, 2016.

[38] "Lightweight directory access protocol," https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol, 2016.

[39] "Role-based access control," https://en.wikipedia.org/wiki/Role-based_access_control, 2016.

[40] "Apache directory," http://directory.apache.org/apacheds/, 2016.

[41] "Apche tomcat," http://tomcat.apache.org/, 2016.

[42] "Prometheus exporter," https://prometheus.io/docs/instrumenting/exporters/, 2015.

[43] "Nagios," https://www.nagios.org/, 2016.

[44] "Nagiosexhange," https://exchange.nagios.org/, 2016.

[45] "Activiti eclipse designer," http://docs.alfresco.com/4.1/tasks/wf-install-activiti-designer.html, 2016.

[46] "Workflow engine jbpm," http://www.jbpm.org/, 2016.

[47] "Workflow engine bonitasoft," http://www.bonitasoft.com/, 2016.

[48] "Activiti execution listeners," http://www.activiti.org/userguide/#executionListeners, 2016.

[49] "Publish/subscibe system zeromq," http://zguide.zeromq.org/, 2016.

[50] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benbernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro, "A survey on service quality description." *ACM Comput. Surv.*, vol. 46, no. 1, p. 1, 2013.

[51] K. Kritikos and D. Plexousakis, "Owl-q for semantic qos-based web service description and discovery." in *SMRR*, ser. CEUR Workshop Proceedings, T. D. Noia, R. Lara, A. Polleres, I. Toma, T. Kawamura, M. Klusch, A. Bernstein, M. Paolucci, A. Leger, and D. L. Martin, Eds., vol. 243. CEUR-WS.org, 2007.

[52] "Kairosdb fast time series database on cassandra," https://kairosdb.github.io/, 2016.

[53] "Apache cassandra database," http://cassandra.apache.org/, 2016.

[54] "Nagios json cgi," https://labs.nagios.com/2014/06/19/exploring-the-new-json-cgis-in-nagios-core-4-0-7-part-1/, 2016.

[55] "Gson library," https://github.com/google/gson, 2016.

[56] I. K. Kim, W. Wang, and M. Humphrey, "Pics: A public iaas cloud simulator." in *CLOUD*, C. Pu and A. Mohindra, Eds. IEEE, 2015, pp. 211–220.

[57] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "Cloudsimsdn: Modeling and simulation of software-defined cloud data centers." in *CCGRID*. IEEE Computer Society, 2015, pp. 475–484.

[58] "Amazon elastic compute," https://aws.amazon.com/ec2/, 2016.

[59] "Google cloud platform," https://cloud.google.com/, 2016.

[60] "Amazon cloudwach," https://aws.amazon.com/cloudwatch/, 2016.

[61] "Kairos db java client library," https://kairosdb.github.io/docs/build/html/KairosClient.html, 2016.

[62] (2008) Selenium.

[63] "Selenium grid for rc and webdriver," https://github.com/SeleniumHQ/selenium/wiki/Grid2, 2016.

Appendices

Appendix A

Metric Tables

Quality Attributes	**Quality Metrics**	**Values**
Networking	Mean Packet Loss Freq	0.1125
	Max Connection Error Rate	0.0
	Response Time	60.72 milli
	Packet round Trip Transfer	0.077 s
Bandwidth	Max Incoming Bandwidth	13.64 Mb/s
	Max Incoming Bandwidth	0.0
CPU Utilization	Type of hypervisor	1.0
	CPU Usage	10.5 %
	VirtualCPUs	2
	Multi-CPU Overall Avg Load	13.61
	Multi-CPU Overall Max Load	13.64
	Multi-CPU Overall Min Load	0.2 %
Hard Disk Performance	Mean Hard Disk Write	0.2772 s
	Max Hard Disk Write	0.2728 s
	Min Hard Disk Write	0.2712 s
	Mean Hard Disk Read	0.172 s
	Max Hard Disk Read	0.1902 s
	Min Hard Disk Read	0.0123 s
	Avg Hard Disk Utilization	87,4125 %

Table A.1: Infrastructure quality metric values

Quality Dimension	**Quality Attribute**	**Quality Metrics**	**Values**
Performance	Response Time	Request Completion Time(avg)	2275.25 milli
		Raw Response Time (avg)	220 milli
		Execution Time (avg)	0.032 milli
		Process Time (avg)	0.02 milli
		Delay Time (avg)	0.012 milli
		Answer Delay Time (avg)	0.034 milli
		Network Latency (avg)	60.72 milli
		Maximum Response Time	282.51 milli
	Throughput	Average Request Number	50.15 h
		Maximum Throughput	52.22 h
Stability	Availability	Down Time	12%
		Availability	88%
	Reliability	MTTF	0.3 milli
		MTBF	1.29 milli
		Fidelity	0.72
		Succesability (avg)	88 %
		NResM	736 h
		Number Of Request Messages	720 h
		NReqM	0.0

Table A.2: Service quality average metric values

Quality Dimension	**Quality Metrics**	**Values**
Time	WPT	449632 milli
	WET	449134 milli
	WDT	813 milli
	WTDT	219 milli
	OWET	449632
	TSDT	2.4
	TDT	2.34567
	PDT	203091.5 milli
	TExT (avg)	40727.42
Cost	WC	9232.80
	WEC	12823
	MWIC	111.43
	RTC	170.72
	RCC	12577
	TC (avg)	450.72
	TEC (avg)	68.4
	SuC (avg)	5.6
	Tdc(avg)	4.2
	TExC(avg)	444.42
	DLC(avg)	256.37
	McN(avg)	262.58
	ARC(avg)	320
Reliability	WA	88.2 h %
	WF	0.72 h %
	TWFT	71.82 h
	WFT	71.82 h
	TFR (avg)	40.2 h
	TF (avg)	0.85 h
	TA (avg)	75.8 h
	MTTR	1.2 milli h
	MTBF	1.4 h
	TotalUpTime (avg)	56.65 h
	NumberOfBreakDowns(avg)	36 h
Security	TFAu	10
	TTAAuth	35
	TVAu	5
	TAuS	85.8

Table A.3: Workflow quality metric values