

Zoltán Ádám Mann
Volker Stolz (Eds.)

Communications in Computer and Information Science

824

Advances in Service-Oriented and Cloud Computing

Workshops of ESOC 2017
Oslo, Norway, September 27–29, 2017
Revised Selected Papers



Springer

Communications in Computer and Information Science

824

Commenced Publication in 2007

Founding and Former Series Editors:

Alfredo Cuzzocrea, Xiaoyong Du, Orhun Kara, Ting Liu, Dominik Ślęzak,
and Xiaokang Yang

Editorial Board

Simone Diniz Junqueira Barbosa

*Pontifical Catholic University of Rio de Janeiro (PUC-Rio),
Rio de Janeiro, Brazil*

Phoebe Chen

La Trobe University, Melbourne, Australia

Joaquim Filipe

Polytechnic Institute of Setúbal, Setúbal, Portugal

Igor Kotenko

*St. Petersburg Institute for Informatics and Automation of the Russian
Academy of Sciences, St. Petersburg, Russia*

Krishna M. Sivalingam

Indian Institute of Technology Madras, Chennai, India

Takashi Washio

Osaka University, Osaka, Japan

Junsong Yuan

Nanyang Technological University, Singapore, Singapore

Lizhu Zhou

Tsinghua University, Beijing, China

More information about this series at <http://www.springer.com/series/7899>

Zoltán Ádám Mann · Volker Stolz (Eds.)

Advances in Service-Oriented and Cloud Computing

Workshops of ES OCC 2017
Oslo, Norway, September 27–29, 2017
Revised Selected Papers

Editors

Zoltán Ádám Mann
University of Duisburg-Essen
Essen
Germany

Volker Stolz
Western Norway University
of Applied Sciences
Bergen
Norway

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-319-79089-3 ISBN 978-3-319-79090-9 (eBook)
<https://doi.org/10.1007/978-3-319-79090-9>

Library of Congress Control Number: 2018939618

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains the joint workshop proceedings of the events co-located with the 6th European Conference on Service-Oriented and Cloud Computing (ESOCC), held in Oslo, Norway, September 27–29, 2017. ESOCC 2017 was organized by the University of Oslo, Norway.

On the first day of the conference, the following workshops were held:

- The First International Workshop on Business Process Management in the Cloud (BPM@Cloud)
- The Third International Workshop on Cloud Adoption and Migration (CloudWays)
- The EU Projects Track

Each workshop submission was reviewed by at least three Program Committee members, and the authors contributed revised articles to this volume of proceedings, taking additional feedback during the workshop into account. Here, a brief description of each workshop is given.

The BPM@Cloud 2017 workshop focused on business process management in the cloud. Organizations are continuously thinking of moving to the cloud in order to reduce costs as well as allow a more flexible provisioning of their business processes (BPs) and services. As they do not possess the appropriate expertise, there is a need to support them via platforms able to realize the respective methods, techniques, and algorithms that concentrate on checking which parts of their BPs should be moved to the cloud, bridging the gap between the business and IT level, and supplying assistance to the four main life cycle activities of BP design, allocation, execution, and evaluation in the cloud. In this respect, the notion of BP as a service (BPaaS), i.e., a Web-based business process that runs in the cloud, arises, which is projected to become quite profitable for the organizations participating in its value chain. As such, BPaaS not only caters for migrating existing BPs in the cloud but can also become a novel exploitation product in the cloud stack that will further boost the adoption of cloud computing. The BPM@Cloud 2017 workshop brought together experts on business process management (BPM) and cloud computing from both academia and industry. It became a medium for thorough discussion and collaboration between its participants. It enabled its presenters to disseminate work that better promotes the notion of BPaaS while also fostering the multidisciplinary collaboration between different research areas in cloud computing and BPM to realize this notion. New challenges were also identified, which can direct the research to be conducted over BPM in the cloud in the near future. The first part of this volume includes all the technical papers of BPM@Cloud 2017.

The CloudWays 2017 workshop focused on cloud adoption and migration. Regardless of the benefits of cloud computing, many organizations still rely on business-critical applications in the form of legacy systems that have been developed over a long period of time using traditional development methods. Despite often serious maintainability issues, (on-premise) legacy systems are still crucial as they support core

business processes. Therefore, migrating legacy systems toward cloud-based platforms allows organizations to leverage their existing systems deployed and provided (using publicly available resources) as scalable cloud services. The CloudWays 2017 workshop brought together cloud migration experts from both academia and industry: to promote discussions and collaboration among participants; to help disseminate novel cloud adoption, migration, and software architecture practices and solutions; and to identify future cloud architecture challenges and dimensions. The second part of this volume includes all the technical papers of CloudWays 2017.

The EU Projects Track 2017 aimed at presenting the major running European-funded projects highlighting the main industrial and academic trends in terms of research and innovation in service-oriented and cloud computing-related domains. The third part of this volume includes all papers of the EU Projects Track 2017.

The contributions of the main conference are published as the 6th IFIP WG 2.14 European Conference ESOC 2017 proceedings in LNCS vol. 10465 by Springer, 2017.

As workshop organization chairs, we would like to thank the individual workshop chairs for their efforts in publicizing the events and succeeding in bringing together participants and authors with contributions reflecting the state of the art. We would also like to thank all authors who presented their work at the events and are contributing here in this volume.

We are also grateful to the local organization team in Oslo that took care of most logistical aspects of organizing such an event, and to the general chair of ESOC, Einar Broch Johnsen, for giving us the opportunity to hold these events together with the conference.

January 2018

Zoltán Ádám Mann
Volker Stolz

Contents

BPM@Cloud

| | |
|---|----|
| Towards PaaS Offering of BPMN 2.0 Engines: A Proposal for Service-Level Tenant Isolation | 5 |
| <i>Majid Makki, Dimitri Van Landuyt, and Wouter Joosen</i> | |
| CEP-Based SLO Evaluation | 20 |
| <i>Kyriakos Kritikos, Chrysostomos Zeginis, Andreas Paravoliasis, and Dimitris Plexousakis</i> | |
| Towards Business-to-IT Alignment in the Cloud | 35 |
| <i>Kyriakos Kritikos, Emanuele Laurenzi, and Knut Hinkelmann</i> | |

CloudWays

| | |
|---|-----|
| Engineering Cloud-Based Applications: Towards an Application Lifecycle | 57 |
| <i>Vasilios Andrikopoulos</i> | |
| A Cloud Computing Workflow for Managing Oceanographic Data | 73 |
| <i>Salma Allam, Antonino Galletta, Lorenzo Carnevale, Moulay Ali Bekri, Rachid El Ouahbi, and Massimo Villari</i> | |
| An Ontology-Based Architecture for an Adaptable Cloud Storage Broker. | 86 |
| <i>Divyaa Manimaran Elango, Frank Fowley, and Claus Pahl</i> | |
| Cloud-Native Databases: An Application Perspective | 102 |
| <i>Josef Spillner, Giovanni Toffetti, and Manuel Ramírez López</i> | |
| Testing and Comparing the Performance of Cloud Service Providers Using a Service Broker Architecture | 117 |
| <i>Divyaa Manimaran Elango, Frank Fowley, and Claus Pahl</i> | |
| TOSKER: Orchestrating Applications with TOSCA and Docker | 130 |
| <i>Antonio Brogi, Luca Rinaldi, and Jacopo Soldani</i> | |

EU Projects

| | |
|--|-----|
| Secure Data Processing in the Cloud | 149 |
| <i>Zoltán Ádám Mann, Eliot Salant, Mike Surridge, Dhouha Ayed, John Boyle, Maritta Heisel, Andreas Metzger, and Paul Mundt</i> | |

| | |
|---|------------|
| DITAS: Unleashing the Potential of Fog Computing to Improve Data-Intensive Applications | 154 |
| <i>Pierluigi Plebani, David Garcia-Perez, Maya Anderson, David Bermbach, Cinzia Cappiello, Ronen I. Kat, Achilleas Marinakis, Vretos Moulos, Frank Pallas, Barbara Pernici, Stefan Tai, and Monica Vitali</i> | |
| HyVar: Scalable Hybrid Variability for Distributed Evolving Software Systems | 159 |
| <i>Thomas Brox Røst, Christoph Seidl, Ingrid Chieh Yu, Ferruccio Damiani, Einar Broch Johnsen, and Cristina Chesta</i> | |
| Enhancing Big Data Application Design with the DICE Framework | 164 |
| <i>Giuliano Casale and Chen Li</i> | |
| Developing, Provisioning and Controlling Time Critical Applications in Cloud | 169 |
| <i>Zhiming Zhao, Paul Martin, Andrew Jones, Ian Taylor, Vlado Stankovski, Guadalupe Flores Salado, George Suci, u, Alexandre Ulisses, and Cees de Laat</i> | |
| MIKELANGELO: Micro Kernel virtualizAtion for hiGh pErformance cLoud and HPC Systems | 175 |
| <i>Nico Struckmann, Yosandra Sandoval, Nadav Har'El, Fang Chen, Shiqing Fan, Justin Činkelj, Gregor Berginc, Peter Chronz, Niv Gilboa, Gabriel Scalosub, Kalman Meth, and John Kennedy</i> | |
| BASMATI: Cloud Brokerage Across Borders for Mobile Users and Applications | 181 |
| <i>Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Konstantinos Tserpes, John Violos, Young-Woo Jung, Ganis Zulfa Santoso, Jorn Altmann, Jamie Marshall, Eric Pages, and Myoungjin Kim</i> | |
| C4E: Cloud Brokering Platform for Federated Services Aimed at European Public Administrations | 187 |
| <i>Antonino Galletta, Oliver Ardo, Antonio Celesti, Peter Kissa, and Massimo Villari</i> | |
| Author Index | 193 |

BPM@Cloud

Preface of BPM@Cloud 2017

To reduce costs as well as allow a more flexible provisioning of their business processes (BPs) and services, organizations are continuously thinking of moving to the cloud. However, most of them do not have the appropriate expertise for performing this move. As such, there is a need for platforms which are able to realize the respective methods, techniques, and algorithms that provide the appropriate cloud-based support level to these organizations. This support level should enable organizations to check which parts of their business processes should be moved to the cloud as well as facilitate bridging the gap between the business and IT level. Moreover, it should assist in the allocation, publishing, execution, monitoring, evaluation, and adaptive provisioning of these BPs, thus catering for their appropriate management based on the four main life cycle activities of BP design, allocation, execution, and evaluation. By moving and offering BPs in the cloud, the notion of BP as a service (BPaaS) is realized, which is projected to become quite profitable for the organizations participating in its value chain that can play the role of brokers, BPaaS management platform operators, or software developers. Thus, BPaaS not only caters for migrating existing BPs in the cloud but can also become a novel exploitation product in the cloud stack that will further boost the adoption of cloud computing.

This was the first edition of this workshop — the First International Workshop on Business Process Management in the Cloud (BPM@Cloud 2017) — which was held in Oslo, Norway, on September 27, 2017, as an ESOC satellite event. The main goals of the workshop were the following: (a) to bring together experts on business process management (BPM) and cloud computing from both academia and industry; (b) to become a medium for thorough discussion and collaboration between the workshop participants; (c) to enable the dissemination of work that better promotes the notion of BPaaS; (d) to foster the multidisciplinary collaboration between different research areas in cloud computing and BPM to realize the notion of BPaaS; (e) to identify new challenges that can direct the research to be conducted over BPM in the cloud in the near future. New versions of the workshop are planned to be organized in the next few years, preferably as satellite events of ESOC.

In this first edition of this workshop, the following three full papers were accepted for presentation.

The first paper “Toward PaaS Offering of BPMN 2.0 Engines: A Proposal for Service-Level Tenant Isolation” by Majid Makki, Dimitri Van Landuyt, and Wouter Joosen explores the main security issues related to the offering of BPMN2-compliant workflow engines in multi-tenant PaaS environments and proposes a service-level tenant isolation framework to address them by also discussing the technical feasibility of its implementation.

The second paper “CEP-Based SLO Evaluation” by Kyriakos Kritikos, Chrysostomos Zeginis, Andreas Paravoliassis, and Dimitris Plexousakis discusses the main issues involved in cross-layer cloud application monitoring and proposes a complex event processing

(CEP) SLO evaluation framework to support the rapid and scalable identification of complex event patterns that signify complex problematic situations which can be addressed via the triggering of cross-layer adaptation workflows.

The third paper “Toward Business-to-IT Alignment in the Cloud” by Kyriakos Kritikos, Emanuele Laurenzi, and Knut Hinkelmann discusses the main issues involved in the design of BPaaS services and proposes a novel semantic framework which is able to align high-level, domain-specific business processes to service-based technical workflows. The novel features of that framework include a questionnaire-based approach for the discovery of services matching the user requirements at the business level as well as a workflow concretization method which takes into account both technical requirements as well as the message compatibility between the discovered services in order to find the most optimal service agglomeration that realizes the functionality of the technical workflow.

In addition to the presentation of the accepted papers, an invited talk titled “Business Processes and Smart Devices — A Marriage of Convenience?” was jointly organized with participants of the Cloudways workshop focusing on the challenges and perspectives with process modelling in the cloud, looking specifically also at edge and IoT as a context. The presentation was given by Prof. Pierluigi Plebani from the Politecnico di Milano, Italy.

We would like to thank all authors, members of the Program Committee, and workshop participants, as their involvement was indispensable for the success of the workshop. A special credit goes to the Information System Laboratory of ICS-FORTH as well as to the members of the Horizon 2020 Cloudsocket European project.

Kyriakos Kritikos

Organization

Program Committee

| | |
|------------------------|--|
| Vasilios Andrikopoulos | University of Groningen, The Netherlands |
| Claudia-Melania Chituc | Eindhoven University of Technology, The Netherlands |
| Marco Comuzzi | Unist, South Korea |
| Schahram Dustdar | Vienna University of Technology, Austria |
| Vincent Emeakaroha | University College Cork, Ireland |
| Ana Juan Ferrer | ATOS, Spain |
| Giancarlo Fortino | University of Calabria, Italy |
| Stella Gatzju Grivas | University of Applied Sciences Northwestern – FHNW, Switzerland |
| Farideh Heirari | Eindhoven University of Technology, The Netherlands |
| Knut Hinkelmann | University of Applied Sciences Northwestern – FHNW, Switzerland |
| Christian Janiesch | University of Würzburg, Germany |
| Dimka Karastoyanova | Kuhne Logistics University, Germany |
| Massimo Mecella | Sapienza - Università di Roma, Italy |
| Jan Mendling | Vienna University of Economics and Business, Austria |
| Adrian Mos | XEROX Research, France |
| Oscar Pastor | Polytechnic University of Valencia, Spain |
| Barbara Pernici | Politecnico di Milano, Italy |
| Pierluigi Plebani | Politecnico di Milano, Italy |
| Dimitris Plexousakis | FORTH, Greece |
| Barbara Re | University of Camerino, Italy |
| Barbara Weber | Technical University of Denmark, Denmark |
| Stefan Wesner | University of Ulm, Germany |
| Robert Woitsch | BOC, Austria |



Towards PaaS Offering of BPMN 2.0 Engines: A Proposal for Service-Level Tenant Isolation

Majid Makki^(✉), Dimitri Van Landuyt, and Wouter Joosen

imec-DistriNet, KU Leuven, 3001 Heverlee, Belgium
{majid.makki,dimitri.vanlanduyt,wouter.joosen}@cs.kuleuven.be

Abstract. Business processes modeling and management solutions provide powerful abstraction mechanisms for the control flow of complex, task-driven applications, and as such allow for better alignment with business-related concerns. Despite the existence and wide adoption of standardized business process management languages such as WS-BPEL and BPMN 2.0, workflow engines in current Platform-as-a-Service (PaaS) offerings are in practice more restricted, in part for reasons such as vendor lock-in, but also due to restrictions of multi-tenant environments.

In this paper, we explore the main security-related problems caused by offering BPMN2-compliant workflow engines in a multi-tenant PaaS environment, particularly focusing on threats caused by misbehaving tenants and the lack of proper tenant isolation. In addition, we propose a service-level tenant isolation framework that allows PaaS offerings to support workflow engines which comply with the BPMN 2.0 standard, and we discuss the technical feasibility of implementing this framework using Java technologies such as OSGi and the Resource Consumption Management API (JSR-284).

Keywords: Platform-as-a-Service (PaaS) · Workflow engines
Multi-tenancy · Untrusted code · Tenant isolation

1 Introduction

Platform-as-a-Service (PaaS) is a category of cloud computing services where the execution platform is offered to software teams for facilitating the development, deployment and maintenance of applications [1,2]. When optimized resource utilization is among the main goals, different applications may share a single installation of the execution platform in a multi-tenant fashion. Workflow engines, in charge of executing business processes, can be part of a PaaS offering and, thus, shared among multiple tenant applications.

Renowned workflow engines in PaaS offerings, such as Amazon SWF [3] and Fantasm in Google App Engine [4], incur a high degree of vendor lock-in and are limited in functionality and suboptimal vis-à-vis utilization of resources. Since these engines have their own custom (i.e. non-standard) workflow modeling languages, the application will be, *de facto*, locked-in by the PaaS provider due to

high cost of porting (cf. [5]). In addition, some features, such as human tasks or advanced event handling mechanisms which are commonly used in state-of-the-art business process automation (cf. [6]), are not supported out of the box. Supporting such features requires quite some ad-hoc engineering effort by the application developers. Furthermore, despite sharing the execution environment of the workflow engine between multiple tenant applications, these solutions require separate environments for executing workflow tasks of each distinct tenant application. The latter decreases resource efficiency whose maximization is a principal goal of cloud computing [7].

These problems can be solved by offering workflow engines that comply with the Business Process Modeling and Notation 2.0 (BPMN 2.0) [8] specification which, next to the Business Process Execution Language (BPEL) [9], is the standard increasingly being adopted in practice. The standardized nature of such engines increases the portability of applications developed using them. In addition, thanks to accumulated experience of decades which is behind BPMN 2.0, these engines do not lack necessary and mainstream functional features. Furthermore, these engines are capable of executing workflow tasks in the same execution environment as the engine itself. Thanks to this capability, resources are utilized more efficiently.

However, PaaS offering of BPMN2-compliant engines causes certain security threats which necessitates specific protection measures well beforehand. The principal source of threats is the untrusted tenant-provided code of workflow tasks that will be executed in an execution environment shared between the PaaS provider and multiple, possibly competing, tenants (cf. [10]). For instance, conflicting access to IO resources is possible. As an alternative example, tenants may exhaustively consume resources such as memory and bring down the service entirely.

The state-of-the-art protection mechanism against such threats is OS-level virtualization, i.e. hypervisors [11] or containers [12], where granularity-level of tenant isolation is, as shown in Fig. 1(a), that of Operating System (OS) processes. This requires having at least one active OS process for each tenant which implies that quite some resources are reserved even if the tenant application does not impose any load. Moreover, OS-level virtualization is not sufficient for all functionalities of BPMN2-compliant engines because they execute some workflow tasks within the same OS process as the engine itself¹ which requires sharing a single OS process between the PaaS provider and tenants. For more efficient utilization of resources and being compatible with the nature of BPMN2-compliant engines, tenant isolation has to take place at a higher level in the computational stack. As depicted by Fig. 1(b), this is the level where the service itself is implemented.

This work-in-progress paper proposes a *service-level* tenant isolation framework for enabling PaaS offering of BPMN 2.0 engines based on Java technologies. We formulate a concrete research problem by analyzing the BPMN 2.0 specification and widely-used BPMN2-compliant engines. Furthermore, given the fact that the *absolute* majority of BPMN2-compliant engines are Java-based [13],

¹ This is required by the BPMN 2.0 specification for some types of tasks.

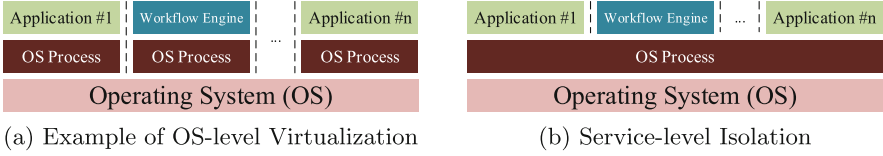


Fig. 1. OS-level virtualization requires having separate OS processes for each tenant application and the workflow engine while service-level isolation allows running all code inside a single OS process.

we present an initial outline of a solution based on Java-related technologies. The proposed solution takes threads as units of isolation for executing untrusted code of tenant applications to overcome the aforementioned insufficiency of the OS-level virtualization approach and the suboptimal resource utilization thereof. The technical feasibility of the solution is shown by explaining how existing technologies, such as OSGi [14] and the Java Resource Consumption Management API (JSR-284 [15]), enable its implementation.

The rest of this paper is structured as follows. Section 2 analyzes the research problem. Section 3 presents the solution outline along with remarks on its technical feasibility. Section 4 briefly contrasts this proposal with related work. Finally, Sect. 5 concludes the paper.

2 Problem Statement

This section analyzes the most compelling security threats caused by the PaaS offering of BPMN 2.0 engines and formulates the research problem as a number of concrete requirements.

2.1 Security Threat Analysis

We have systematically analyzed and prioritized the security threats using the **STRIDE**² threat model [16, 17]. The most problematic security threats, insofar as this paper is concerned, are related to two *core* features of BPMN 2.0 namely **Script Task** and **Service Task** activity types. The former is required by the standard to be “executed by a business process engine” [8]. This implies that the same OS process running the engine is responsible for executing the **Script Task**. While the standard does not require **Service Task** activities to be executed by the same Operating System (OS) process running the engine, most well-known and enterprise-ready BPMN 2.0 engines, such as jBPM [18] and Activiti [19], allow defining **Service Task** activities that are executed within the same OS process running the engine. Retaining this additional feature is

² The acronym stands for six threat categories namely **S**poofing, **T**ampering with **D**ata, **R**epudiation, **I**nformation Disclosure, **D**enial of Service and **E**levation of Privilege.

essential for avoiding the overhead of remote service invocation, e.g. (de-)serialization and network delay.

In a multi-tenant context of a PaaS offering, the code of **Script Task** and **Service Task** activities belong to untrusted tenant applications using the engine. Executing untrusted code of tenants in the same OS process running the engine incurs different types of security threats (cf. [10,20,21]). Tampering with Data, Information Disclosure, Denial of Service and Elevation of Privilege are identified as the most important threat categories in this specific context and are discussed below.

Tampering with Data. Since tenant-provided code may access IO resources, one tenant application may modify another tenant’s data stored on a shared device. In addition, a tenant application may modify the value of in-memory object references belonging to or shared with other tenants.

Information Disclosure. A tenant application may read another tenant’s data by, e.g., listening on a network port belonging to the other tenant. Similarly, a tenant application may access in-memory object references belonging to or shared with other tenants.

Denial of Service. One tenant application may disrupt the PaaS offering entirely either by *using up* computational resources or by misusing part of the API shared among all tenants. The resources of concern are CPU cycles, memory space and IO bandwidth (both storage and network). Misuse of shared API can be either in form of killing the OS process hosting the service or in form of locking shared objects *indefinitely*.

Elevation of Privilege. By creating a thread which is not under control of the framework, one tenant application may increase its privileges and act without constraints imposed by the framework.

2.2 Requirements

Since these threats are caused by code running within a single OS process, protection against them has to take place inside that OS process as well. Therefore, a *service-level* tenant isolation framework is needed which fulfills the following functional requirements:

- FR1: The framework should guarantee that no tenant application may access objects or primitive values belonging to other tenants.
- FR2: The framework has to guarantee that shared system objects and references accessible for tenant applications can neither be locked indefinitely nor be modified by any of them.
- FR3: The creation of threads has to be entirely mediated by the framework.
- FR4: Permission of killing the OS process has to be denied for all tenant applications.
- FR5: The framework should check tenant permissions before granting access to any IO resource (e.g. a file on storage device or a network port).

- FR6: An upper limit has to be put on CPU usage, memory consumption and IO bandwidth (both storage and network) on a per-tenant basis.
- FR7: Upper limit imposition on resource consumption has to be so flexible that resource utilization maximizes. In other words, if there are unused resources that can be allocated safely, the framework should let tenant code exceed the limits to some extent.

In addition, fulfillment of the above functional requirements should respect the following quality requirements:

- QR1: All tenant isolation measurements should be enforced transparently by the framework. In other words, code of tenant applications has to be entirely decoupled from the tenant isolation framework.³
- QR2: The relative performance overhead of the framework compared to OS-level virtualization tactics (cf. [11,12]) is required to be in an acceptable margin.

3 Service-Level Tenant Isolation

This section outlines a service-level tenant isolation framework as a solution for fulfilling the above requirements and shows technical feasibility of the framework. Section 3.1 presents the framework architecture conceptually. Section 3.2 elaborates on partial fulfillment of FR1 while Sect. 3.3 supplements it and discusses static code restriction which is required for fulfillment of FR2 and FR3. Supplementary measures for fulfillment of FR2 and FR3 are presented in Sect. 3.4 while Sect. 3.5 deals with permission checking mechanism which is required for FR2, FR3, FR4 and FR5. Finally, Sect. 3.6 explains how the framework realizes FR6 and FR7. The qualitative requirements are taken into account orthogonally throughout this section.

3.1 Overall Architecture

Figure 2 shows the principal components running within a single OS process in three layers: (i) the bottom layer is the Java execution platform and the components it provides, (ii) the middle layer is where the service components of the PaaS offering, including components of the tenant isolation framework, sit, and (iii) the top layer consists of code of tenant applications built upon the PaaS offering.

The tenant isolation framework consists of seven main components which are introduced gradually throughout this section. The two front-end components of the framework which are directly used by tenants are **Deployment Service** and **Workflow Execution Service**. The former is responsible for deploying tenant applications into the PaaS environment. The latter is responsible for starting new workflow executions or continue/monitor/abort existing ones by running the code of tenant applications in isolation from other tenants.

³ This is required for portability of tenant applications to other instances of the same BPMN 2.0 engine where the tenant isolation framework is not used.

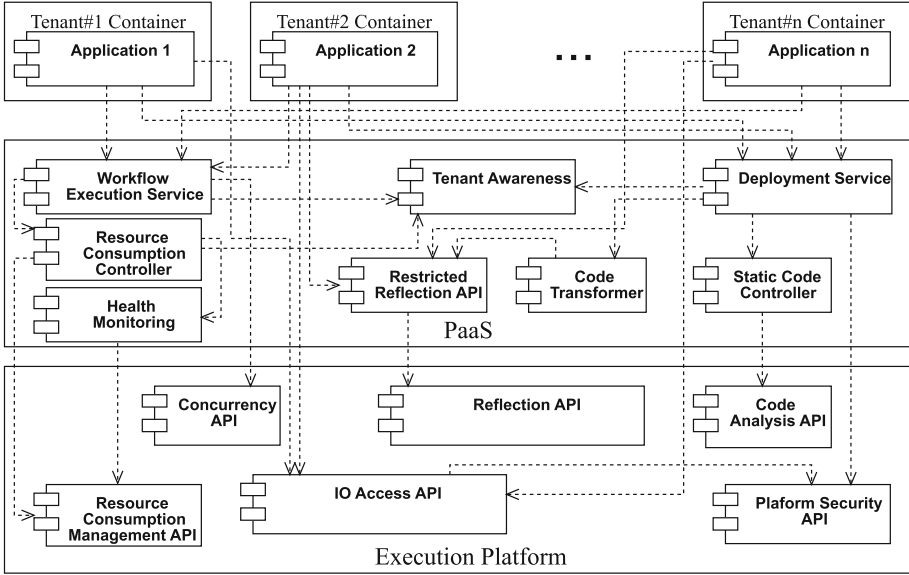


Fig. 2. Building blocks of the framework in relation to tenant applications and the execution platform

For imposing isolation measures, the **Workflow Execution Service**, using an existing BPMN2-compliant engine such as jBPM [18], runs the untrusted code of tenant applications in separate threads and guarantees that each active thread is associated with only one tenant application at a time. Once each thread finishes its job, it can be reused for other tenant applications using a thread pool provided by the Java Concurrency API. The **Workflow Execution Service** leverages upon the **Concurrency API** of the Java execution platform for handling threads. Furthermore, it uses the **Tenant Awareness** component of the framework which is responsible for keeping track of associations between threads and tenants. Thus, the main cornerstone of the proposed solution is taking threads as units of tenant isolation just as OS-level virtualization tactics take OS processes as units of tenant isolation.

3.2 Tenant Containers

In order to dedicate a separate referencing space for objects of each tenant application (cf. FR1), the **Deployment Service** deploys each application in a distinct tenant container. As opposed to containers of OS-level virtualization approach, containers shown in Fig. 2 are managed inside a single OS process. OSGi [14] bundles provide exactly this containerization functionality. OSGi loads each bundle using a distinct Java classloader and sets the bootstrap classloader, which is responsible for loading core Java classes, as the parent of each bundle

classloader. Hence, the code of each bundle can access fields of its own classes and *static* fields of core Java classes.

By containing the code of each tenant application in a separate OSGi bundle, the FR1 requirement will be partially fulfilled. For complete fulfillment of FR1, cross-bundle communication between tenant application bundles has to be forbidden which is the topic of next section along with realization of FR2 and FR3.

3.3 Static Code Restriction

Access of tenant applications to other classes and interfaces has to be restricted for fulfillment of FR1, FR2 and FR3. API restriction is required both statically and dynamically. Static restriction takes place only once at deployment time by the **Static Code Controller** (cf. Fig. 2). This is done in multiple stages using code analysis facilities provided by the Java platform and tools built upon it.

Cross-bundle Communication. Each OSGi bundle declares the list of classes it imports from other bundles. This is used for cross-bundle communication. By imposing limits on this list, the **Static Code Controller** guarantees that no cross-bundle communication is possible between two tenant applications and, thus, completes the realization of FR1.

Blacklist. One of the main elements of **Static Code Controller** is **BlacklistService** which maintains a list of classes, methods and fields that tenant applications are not allowed to use.

Given the structure of classloaders in OSGi, security vulnerabilities pertaining to shared object locks and modifications (cf. FR2) are caused by four Java programming constructs related to classes loaded by the bootstrap classloader: (i) *static* field declarations, (ii) reference updates on *static* fields, (iii) changing state of objects referenced by *static* fields, (iv) *static synchronized* methods, and (v) *synchronized* blocks locking *static* fields [10,20,21]. The **BlacklistService** searches for occurrences of these constructs in all classes loaded by the bootstrap classloader using the Java source code querying facilities provided by the Spoon library [22] as well as call graph construction and reference analysis (a.k.a. points-to analysis) mechanisms of the Soot framework [23]. A call graph consists of nodes and edges representing Java methods and invocation relationship between them respectively. Reference analysis helps resolving non-static access to objects referenced by *static* fields of concern.

In addition, to further comply with FR3, the **BlacklistService** adds the `java.lang.Thread` class to the blacklist. Furthermore, using the Spoon library, it adds any method in the `java.util.concurrent` package capable of instantiating new threads or intervening in the life-cycle of an existing thread.

Acceptance Policy. The final stage of **Static Code Controller** involves accepting or rejecting the tenant application code. It checks whether the code of a tenant application directly or indirectly deals with blacklisted constructs.

```

SecurityManager sm = System.getSecurityManager();
if (BlacklistService.isBlacklisted(this)) {
    if (sm != null) {
        sm.checkPermission(new
            ReflectPermission("evadeBlacklist"));
    }
}
super.originalMethod(...); // pseudocode

```

Listing 1.1. Restricting access of tenant applications to Java Reflection API.

For instance, it checks whether a blacklisted `static synchronized` is invoked or the constructor of the `Thread` class is used. The first step for doing so is creating a call graph using the Soot framework. Afterwards, the call graph has to be traversed to see if any of the fields, methods or classes in the blacklist is used by the methods included in the call graph. Furthermore, using the reference analysis mechanism provided by Soot, it has to be verified whether objects referenced by *static* fields in the blacklist are modified indirectly (e.g. by means of an intermediate local reference).

3.4 Dynamic Code Restriction

Restricting code of tenant applications statically is not sufficient because all the malicious operations, which can be detected statically, can be done dynamically as well using the `Reflection API`. Therefore, `Restricted Reflection API` should be used by tenant applications instead of the original `Java Reflection API` (cf. Fig. 2). This, however, can reintroduce the vendor lock-in problem that QR1 requires to avoid. Therefore, tenant applications are allowed to use the original `Reflection API` but at deployment-time, the `Deployment Service` asks the `Code Transformer` to transform tenant code such that the original `Reflection API` is replaced by `Restricted Reflection API`. This is feasible and straightforward because the `Restricted Reflection API` has exactly the same package structure and exposes exactly the same API as the original one but with different behavior in some cases. The `Code Transformer` component employs the transformation utilities provided by the Spoon framework [22].

The behavior difference is summarized by Listing 1.1 where `this` either refers to a `Field` object whose `get` method is called, a `Constructor` object whose `newInstance` method is invoked or a `Method` object whose `invoke` method is called. Determining whether the use of these class members are blacklisted for tenant applications, the same `BlacklistService`, which maintains the blacklist, is used. Since the blacklist is prepared once in the entire life-time of the application and kept in memory for subsequent uses, this does not impose a significant performance overhead (cf. QR2).

As shown, `Java SecurityManager` is used to check whether `evadeBlacklist` permission is granted to the OSGi bundle requesting the reflective operation. This permission has to be granted only to trusted bundles, i.e. not to bundles containing code of tenant applications. The `checkPermission` method throws

an `AccessControlException` if the required permission is not granted to the bundle requesting the reflective operation.

The next section elaborates on how the permission checking mechanism of the Java `SecurityManager` works and on how it is employed by the tenant isolation framework.

3.5 Permission Checks

In addition to the above permission checking, the tenant isolation framework enforces permission checks on invocations of `System.exit` method (cf. FR4) and on every IO access (cf. FR5). Permission checks are automatically done by the Java platform itself once the `SecurityManager` is enabled. The role of the framework, hence, is limited to enabling the `SecurityManager` and granting permissions properly.

The `SecurityManager` relies on a mechanism called *stackwalking* and associates a permission set to each *protection domain* (cf. [24,25]). In OSGi, there exists one protection domain for each bundle. The set of permissions of each tenant application is granted to it by the `Deployment Service` at deployment time. Tenant permission sets are, in principle, a combination of `FilePermission` and `SocketPermission` for restricting their access to IO resources which is required by FR5. By denying the `RuntimePermission("exitVM")` to tenant bundles, FR4 is also fulfilled. All other bundles, which do not contain tenant-provided code, are granted `AllPermission`.

When permission p is required, the Java platform `SecurityManager` triggers stackwalking, i.e. tracing the entire method invocation stack which has led to the point of permission checking, and verifies that

$$\forall_{pd \in PD_s} p \in P_{pd} \quad (1)$$

where PD_s is the set of all protection domains involved in the scanned method invocation stack s and P_{pd} is the set of permissions granted to the protection domain pd . This way, the permissions of tenant application for whom permission p has to be checked are taken into account and retrieved from its corresponding protection domain.

Permission checks are not required when the running code is trusted, e.g. when PaaS management and monitoring components are executed. In order to avoid the performance overhead of the `SecurityManager` when it is not needed, the framework enables it only when tenant application code is executed and disables it otherwise. The `TenantWorkflowExecutor`, which is responsible for starting/resuming/aborting a tenant workflow, toggles the `ToggleableSecurityManager` shown in Fig. 3 before and after workflow execution using `enable` and `disable` methods of the latter. These methods change the value of a `ThreadLocal` variable which is used in the `checkPermission` method according to Listing 1.2.

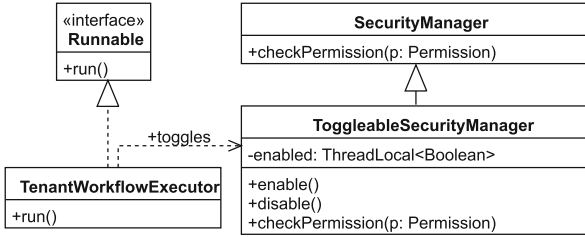


Fig. 3. The framework extends the `SecurityManager` such that it can be enabled only when tenant-provided code is executed. This is done by means of a `ThreadLocal` variable toggled before and after a tenant workflow executes.

```

if (enabled.get()) {
    super.checkPermission(p);
}
  
```

Listing 1.2. Evading permission check when it is not required.

3.6 Resource Consumption Control

Fulfilling FR6 requires associating a service-level agreement (SLA) to each tenant application. Listing 1.3 shows the structure of a tenant SLA. The framework uses the Java Resource Consumption Management API (JSR-284) [15] for imposing limits on resource consumption of each tenant application. On a per-tenant basis, `ResourceMeter` instances are created for each element of the SLA. Meters are notified by the Java platform every time new information about their corresponding resources allocation/release is available. The `Resource Consumption Controller` creates tenant meters only once (the first time they are needed) by consulting the `Tenant Awareness` component which has access to SLAs. Once meters are created, they will be associated with a tenant-specific `ResourceContext`. Before executing the untrusted code of any tenant, the `TenantWorkflowExecutor` associates the tenant meters to the executing thread according to Listing 1.4.

The meters provided by the Java platform are themselves capable of imposing a limit on consumption of IO-related resources. Hence, it is sufficient to choose the right type of meter for each resource type. The `BoundedMeter` is used for `maxOpenFiles`, `maxOpenSockets` and `maxOpenDatagrams` and the `ThrottledMeter` is used for other IO-related resources. Both of these meters are capable of imposing a limit on resource usage. The only difference is that the former is appropriate for cases dealing with absolute numbers (e.g. number of open files) while the latter best suits cases where a rate is involved (e.g. bytes read from file system per seconds).

The meters provided by the Java platform however are not capable of imposing a limit on memory and CPU usage. The best they can do is to notify when the limit is reached. Our framework employs Quasar Fibers [26] instead of original

```
public class TenantSLA {  
  
    private long maxCpuUsage; // CPU nanoseconds per second  
    private long maxMemoryUsage; // bytes  
  
    private int maxOpenFiles;  
    private long maxReadDiskRate; // bytes per second  
    private long maxWriteDiskRate; // bytes per second  
  
    private int maxOpenSockets; // TCP sockets  
    private long maxReadSocketRate; // bytes per second  
    private long maxWriteSocketRate; // bytes per second  
  
    private int maxOpenDatagrams; // UDP datagrams  
    private long maxReadDatagramRate; // bytes per second  
    private long maxWriteDatagramRate; // bytes per second  
  
    // getters and setters  
  
}
```

Listing 1.3. Structure of Tenant SLA.

```
ResourceContextFactory factory = ResourceContextFactory.getInstance();  
ResourceContext rc = factory.lookup(tenantId);  
rc.bindThreadContext(); // binds to the current thread  
// workflow execution code  
rc.unbindThreadContext(); // unbinds from the current thread
```

Listing 1.4. Binding tenant `ResourceContext` to threads before starting workflow execution.

Java threads in order to safely suspend the untrusted code of tenants when they consume too much memory space or CPU time. Since Fiber suspension is done at the level of the JVM rather than that of the OS kernel, regularly suspending them does not impose much overhead. Furthermore, tenant-specific information about consumption level can be used for resuming suspended Fibers whereas execution of original Java threads are left to the OS kernel which does not have any tenant-specific information.

Fiber suspension may take place after specific checkpoints. The memory usage is checked every time a new object is instantiated. Therefore, the bytecode of the `Object` constructor is manipulated to enforce a memory limit checkpoint. Once the responsible `NotifyingMeter` notifies the surpass of memory limit by a specific tenant, a tenant-specific boolean variable will be modified to indicate that the corresponding tenant cannot create new objects anymore. The boolean variable will be consulted by the memory checkpoint inside of the `Object` constructor.

The CPU checkpoints, however, are more widespread. They have to guarantee that tenants cannot evade CPU usage control mechanism. Hence, two types of checkpoints are inserted by bytecode manipulation of both the Java API and the untrusted code of tenant: (i) inside every loop structure and (ii) inside every recursion (be it direct or indirect). Manipulating the Java API is required

because tenants may exploit it by means of method arguments. Similarly, a boolean variable is checked in every checkpoint and if the tenant code has to be suspended due to excess of CPU usage, it will be suspended. Despite the widespread nature of CPU checkpoints, the performance overhead is not expected to be high because under normal circumstances every checkpoint will amount to a simple if statement involving a boolean variable.

FR7 requires managing some types of resources flexibly. Flexible resource consumption means allowing tenants surpassing the limits defined in their SLAs when there are sufficient amount of resources available for allocation. This involves too much risk in case of memory usage, number of open files and number of open sockets because once tenants go beyond their limits on these types of resources, there is no guarantee that the system can push them back to their borders (cf. [10,20,21] for the case of memory usage). However, that risk is not relevant in case CPU usage and IO bandwidth because their consumption is time-dependent by nature and thus can be reclaimed by the system if need be.

The following condition determines whether resource allocation can be done regardless of the fact that a tenant SLA limit is reached:

$$total_r + amount_r \leq l_r \times capacity_r \quad (2)$$

where r is a resource of concern, $total_r$ is the total consumption amount of all tenants before approving the new request, $amount_r$ is the requested amount, $capacity_r$ is the total capacity of the system on r and l_r is the leniency factor between 0 and 1. When leniency factor is set to zero, every tenant will be strictly restricted to its SLA regardless of resource availability, i.e. unallocated amounts. When it is set to one, the unallocated amounts will be used for letting more active tenants going beyond their SLA limits.

The **System Health Monitoring** is consulted component for retrieving the total usage. Capacity is a set by system administrators while total usage is retrieved from the **ResourceContextFactory** provided by the Java platform. The latter does not calculate the total usage every time queried. Instead, it keeps track of total amounts on every resource allocation and release. Decisions of this approver are safe because they are made about resources measured on a per-second basis and surpassing their limits will have no effect beyond the measurement window which is two seconds according to the API. In other words, it is guaranteed that these resources can be throttled back to the limits defined in tenant SLAs once the load on the system increases.

4 Related Work

In this section, we briefly compare this paper with related work.

PaaS Offering of Workflow Engines. Pathirage et al. [27] proposes an architecture for PaaS offering of Apache ODE [28] which is a workflow engine complying with WS-BPEL. Since all activities involving code execution are remote web-services, the security threats that we covered in this paper are not relevant

for that work. However, delegating execution of all untrusted activity code to remote web-services both reduces efficiency of resource utilization and imposes the overhead of remote invocation (e.g. network delay and serialization). Yu et al. [29] claims having enabled jBPM [18], a BPMN2-compliant engine, to be offered as a PaaS. However, the security threats discussed in this paper are overlooked altogether. Amazon SWF [3] and Fantasm on Google App Engine [4] are production ready PaaS offerings of workflow engines. However, applications developed using them highly suffer from vendor lock-in problem in terms of both code and data. Furthermore, resource utilization is sub-optimal due to adoption of OS-level virtualization tactics.

OS-level Virtualization. Similar isolation measures can be imposed by means of containers (cf. [12]) or virtual machines (cf. [11]). However, in case of Java, a separate instance of the JVM should be started for each tenant which reduces efficiency of resource utilization. Furthermore, these solutions are totally insufficient for offering BPMN2-compliant engines as the latter runs `Script Task` and, in some cases, `Service Task` activities in the same OS process as the workflow engine itself.

Java Language Vulnerabilities. Security threats related to running untrusted code in Java threads are discussed in [10,20,21]. These problems are caused by the shared nature of *static* fields, blocking effect of *static synchronized* methods, reference leaks and shared nature of computational resources. Our threat analysis is based on these works and we have proposed ideas for solving some of them and workarounds for some others based on existing technologies.

Application Performance Isolation. There are a number of works dealing with performance isolation for Software-as-a-Service (SaaS) applications [30–33]. While these works deal with SLAs expressed in external properties of an application such as response-time and throughput, the SLAs are defined in system-level terms such as CPU usage. This is because these works do not deal with the issues of running untrusted code in a shared execution environment. Krebs et al. have proposed a framework for determining resource usage based on the aforementioned external properties [34]. However, their solution requires categorization of all possible requests into different groups beforehand which is not feasible in case of PaaS where application requests are not known by the PaaS provider.

5 Conclusion

Business process automation is a common practice supported by a set of mature standards (e.g. BPMN 2.0 and WS-BPEL) and numerous workflow engines that implement these standards. Due to the specific deployment model of multi-tenancy in a Platform-as-a-Service (PaaS) context, full support of these standards requires additional attention to security threats caused by misbehaving tenants. We have presented an outline of a framework for tenant isolation in the context of co-existing business processes of different tenants, and we have discussed its practical feasibility for the Java environment.

Advancing this work fits into our ongoing research on the key trade-offs related to multi-tenancy between resource optimization, customization support (e.g. by means of tenant-provided tasks), security (tenant isolation) and portability of business processes across different cloud providers. We have implemented the permission checking and resource consumption mechanisms of the framework. In follow-up work, we will further implement the code restriction part and evaluate the proposed framework vis-à-vis performance overhead compared to the OS-level virtualization approach and dimension of tenant code restrictions (i.e. determining how limited tenant applications will be in using the Java API given the restrictions imposed by the framework).

Acknowledgement. This research is partially funded by the Research Fund KU Leuven (project GOA/14/003 - ADDIS), the strategic basic research (SBO) project DeCoMAdS, and the MuDCads O&O project.

References

1. Rimal, B.P., Choi, E., Lumb, I.: A taxonomy and survey of cloud computing systems. In: INC, IMS and IDC, pp. 44–51 (2009)
2. Walraven, S., Truyen, E., Joosen, W.: Comparing paas offerings in light of SaaS development. *Computing* **96**(8), 669–724 (2014)
3. AWS: Amazon Simple Workflow Service (Amazon SWF). <https://aws.amazon.com/documentation/swf/>. Accessed 12 June 2017
4. Google: Google App Engine Fantasm. <https://cloud.google.com/appengine/articles/fantasm>. Accessed 12 June 2017
5. Opara-Martins, J., Sahandi, R., Tian, F.: Critical review of vendor lock-in and its impact on adoption of cloud computing. In: 2014 International Conference on Information Society (i-Society), pp. 92–97. IEEE (2014)
6. Ko, R.K., Lee, S.S., Wah Lee, E.: Business process management (BPM) standards: a survey. *Bus. Process Manag. J.* **15**(5), 744–791 (2009)
7. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. *Commun. ACM* **53**(4), 50–58 (2010)
8. OMG: Business Process Model and Notation 2.0. <http://www.omg.org/spec/BPMN/2.0/PDF/>. Accessed 04 Aug 2015
9. OASIS: Web Services Business Process Execution Language. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Accessed 04 June 2016
10. Rodero-Merino, L., Vaquero, L.M., Caron, E., Muresan, A., Desprez, F.: Building safe PaaS clouds: a survey on security in multitenant software platforms. *Comput. Secur.* **31**(1), 96–108 (2012)
11. Li, Y., Li, W., Jiang, C.: A survey of virtual machine system: current technology and future trends. In: 2010 Third International Symposium on Electronic Commerce and Security (ISECS), pp. 332–336. IEEE (2010)
12. Bernstein, D.: Containers and cloud: from LXC to docker to kubernetes. *IEEE Cloud Comput.* **1**(3), 81–84 (2014)
13. Wikipedia: List of BPMN Engines. https://en.wikipedia.org/wiki/List_of_BPMN_2.0_engines. Accessed 05 July 2017
14. OSGi-Alliance: OSGi specification (2012). <https://osgi.org/download/r4v43/osgi-core-4.3.0.pdf>. Accessed 19 April 2017

15. JCP: JSR 284: Resource Consumption Management API. <https://jcp.org/en/jsr/detail?id=284>. Accessed 12 June 2017
16. Microsoft: The stride thread model (2015). [https://msdn.microsoft.com/en-us/library/ee823878\(v=cs.20\).aspx](https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx). Accessed 19 April 2017
17. Shostack, A.: Threat Modeling: Designing for Security. Wiley, New York (2014)
18. RedHat-JBoss: jBPM. <http://www.jbpm.org/>. Accessed 04 June 2017
19. Alfresco: Activiti User Guide. <https://www.activiti.org/userguide/>. Accessed 24 May 2017
20. Czajkowski, G., Daynés, L.: Multitasking without compromise: a virtual machine evolution. ACM SIGPLAN Not. **36**, 125–138 (2001)
21. Herzog, A., Shahmehri, N.: Problems running untrusted services as Java threads. Certification Secur. Inter-Organ. E-Serv. **177**, 19–32 (2004)
22. Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., Seinturier, L.: Spoon: a library for implementing analyses and transformations of Java source code. Softw. Pract. Exp. **46**(9), 1155–1179 (2016)
23. Lam, P., Bodden, E., Lhoták, O., Hendren, L.: The soot framework for Java program analysis: a retrospective. In: Cetus Users and Compiler Infrastructure Workshop (CETUS 2011), vol. 15, p. 35 (2011)
24. Oracle: Java 8 SE platform security. <https://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html>. Accessed 19 April 2017
25. Gong, L., Ellison, G.: Inside Java (TM) 2 Platform Security: Architecture, API Design, and Implementation. Pearson Education, London (2003)
26. Parallel Universe: Quasar. <http://docs.paralleluniverse.co/quasar/>. Accessed 09 July 2017
27. Pathirage, M., Perera, S., Kumara, I., Weerawarana, S.: A multi-tenant architecture for business process executions. In: 2011 IEEE International Conference on Web services (ICWS), pp. 121–128. IEEE (2011)
28. Apache: Apache ode. <http://ode.apache.org/>. Accessed 09 July 2017
29. Yu, D., Zhu, Q., Guo, D., Huang, B., Su, J.: jBPM4S: a multi-tenant extension of jBPM to support BPaaS. In: Bae, J., Suriadi, S., Wen, L. (eds.) AP-BPM 2015. LNBP, vol. 219, pp. 43–56. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19509-4_4
30. Walraven, S., De Borger, W., Vanbrabant, B., Lagaisse, B., Van Landuyt, D., Joosen, W.: Adaptive performance isolation middleware for multi-tenant SaaS. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 112–121. IEEE (2015)
31. Krebs, R., Loesch, M., Kounev, S.: Platform-as-a-service architecture for performance isolated multi-tenant applications. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), pp. 914–921. IEEE (2014)
32. Krebs, R., Momm, C., Kounev, S.: Metrics and techniques for quantifying performance isolation in cloud environments. Sci. Comput. Program. **90**, 116–134 (2014)
33. Lin, H., Sun, K., Zhao, S., Han, Y.: Feedback-control-based performance regulation for multi-tenant applications. In: 2009 15th International Conference on Parallel and Distributed Systems (ICPADS), pp. 134–141. IEEE (2009)
34. Krebs, R., Spinner, S., Ahmed, N., Kounev, S.: Resource usage control in multi-tenant applications. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 122–131. IEEE (2014)



CEP-Based SLO Evaluation

Kyriakos Kritikos¹ , Chrysostomos Zeginis¹ , Andreas Paravoliassis²,
and Dimitris Plexousakis¹

¹ Institute of Computer Science - FORTH, Heraklion, Greece

{kritikos,zegchris,dp}@ics.forth.gr

² Computer Science Department, University of Crete, Heraklion, Greece
csd3031@csd.uoc.gr

Abstract. Modern service-based applications (SBAs) operate in highly dynamic environments where both underlying resources and the application demand can be constantly changing which external SBA components might fail. Thus, they need to be rapidly modified to address such changes. Such a rapid updating should be performed across multiple levels to better deal, in an orchestrated and globally-consistent manner, with the current problematic situation. First of all, this means that a fast and scalable event generation and detection mechanism should exist to rapidly trigger the adaptation workflow to be performed. Such a mechanism needs to handle all kinds of events occurring at different abstraction levels and to compose them so as to detect more advanced situations. To this end, this paper introduces a new complex event processing framework able to realise the respective features mentioned (processing speed, scalability) and have the flexibility to capture and sense any kind of event or event combination occurring in the SBA system. Such a framework is wrapped in the form of a REST service enabling to manage the event patterns that need to be rapidly detected. It is also well connected to other main components of the SBA management system, via a publish-subscribe mechanism, including monitoring and the adaptation engines.

Keywords: Complex event processing · Event pattern · Detection Service

1 Introduction

Due to tough competition, organisations can survive if they can improve their services to exhibit better service levels with less cost. Such organisations need to also possess a smart infrastructure and a dedicated devops team to appropriately re-configure the services offered as well as manually intervene in unanticipated, problematic situations. As such, a lot of effort is spent in maintaining such an infrastructure while an increasing management and operational cost also incurs.

Fortunately, the advent of cloud computing has revolutionised the way resource management is performed. Nowadays, organisations can outsource their infrastructure management to cloud providers that promise to offer infinite,

cheap commodity resources on an on-demand basis. Due to flexible resource management and the capability to scale a cloud-based system, organisations can now optimise their services at the infrastructure level. However, still effort is needed at higher-levels of abstractions. In particular, external SaaS services need to be dynamically selected to realise part of the required functionality while the whole system needs to be adapted.

In the literature, it has been advocated [11] that dynamic SBA adaptation should be performed in a cross-layer manner by also putting in place, as a prerequisite, a suitable monitoring framework. Cross-layer adaptation is needed for various reasons. First, as the service system itself includes multiple levels that must be appropriately controlled. Second, as the individual adaptation at one level can influence, impact or even negate the adaptation results at adjacent levels, leading to a vicious re-adaptation cycle. Cross-layer monitoring is also needed to propagate and aggregate up to higher-levels measurements produced in lower levels so as to cover measurability gaps.

As the glue between monitoring and adaptation, there is a need for a rapid and scalable Service Level Objective (SLO) evaluation framework able to transform measurements to events and subsequently detect event patterns that can lead to performing adaptation actions in the context of adaptation rules. Such a framework should also exhibit suitable accuracy levels by correctly correlating the events occurring based on their metrics and measured objects. It should also be able to detect and correlate events which should map to both the type and instance level in the managed SBA system.

In this work, such a framework has been carefully designed and realised, by conforming to all the aforementioned requirements. In particular, the framework architecture was initially designed by considering principles, such as service-orientation, and by carefully decoupling framework parts subject to scaling. Based on this architecture and the appropriate selection of the right, existing components and tools, a respective framework was then implemented and integrated in our existing SBA monitoring and adaptation framework [21]. Such an integration is loosely coupled as our SLO evaluation framework can be in principle connected to any monitoring and adaptation engine.

The developed framework relies on the CAMEL domain-specific language (DSL), able to capture various aspects in the cloud-based application lifecycle management, including the monitoring and adaptation ones. In particular, this DSL is expressive enough to specify complex event patterns, where each event maps to a metric condition, and associate them with respective sets of adaptation actions that must be triggered to adapt the SBA in a cross-layer manner. CAMEL also covers well the monitoring aspect via its capability to specify how composite metrics are aggregated and to associate metrics with the (e.g., service) component that they measure. As it will be shown, such information is essential to have the ability to correlate events in the context of event pattern detection.

The proposed framework relies on the Esper Complex Event Processing (CEP) engine. This engine is quite scalable with the capability to process thousands or even millions of events. Due to the way our architecture has been

designed, this engine can be scaled when its processing limits are reached, enabling our framework to really scale at those parts where most of the load is directed.

The rest of the paper is structured as follows. The next section provides a use case scenario which is used as a running example across the whole paper, while Sect. 3 reviews the related work. Section 4 provides background information necessary for the comprehension of this paper contribution. Section 5 analyses the proposed framework architecture and supplies some implementation details. Section 6 explains the way the event pattern specification is generated by accounting also on how the events of the pattern should be correlated. Finally, the last section concludes the paper and draws directions for further research.

2 Use Case

The use case, which is used as a running example across the paper, has been drawn from the CloudSocket project¹ which deals with the management of Business Processes (BPs) in the Cloud. This use case concerns the development of a service-based BP as a service (BPaaS), named as “SendInvoice”, which offers the functionality of invoice generation and sending. This BPaaS maps to a supporting BP which can really provide appropriate automation level within a small or medium-sized organisation with respect to the management of invoicing. In this respect, it makes sense to develop and offer this BP in the cloud as the demand for this BP would be quite high.

The “SendInvoice” BPaaS exploits two main services: (a) an external SaaS dedicated to the customer relationship management (CRM) named as YMENS CRM; (b) an internal component for invoice management called “Invoice Ninja” which has been purchased and deployed in the Cloud in an Amazon EC2 VM named as “m1.medium”. These two services are then combined into a technical workflow which is deployed in the cloud and includes tasks that map to certain methods/functionality of these services.

The topology of the initial deployment of the “SendInvoice” BPaaS in the Cloud, as specified also in CAMEL, is depicted in Fig. 1 where both the type and instance levels are shown. As it can be observed, only one instance of the “InvoiceNinja” (software) component, named as “InvoiceNinja_inst1” has been deployed in one instance of the “m1.medium” VM named as “m1.medium_inst1”.

Suppose, now, that the organisation offering the “SendInvoice” BPaaS, i.e., a Cloud Broker, needs to control its execution in order to sustain a suitable service level that has been agreed with any of its customers in the context of an SLA. As the set of customers can grow, the Cloud Broker needs to control the amount of resources dedicated to “Invoice Ninja” as well as have the ability to replace the CRM service when its service level is not any more acceptable. To this end, it specifies the following set of adaptation rules (specified in CAMEL but abstracted away due to space limitation reasons) which scale out “Invoice Ninja”

¹ www.cloudsocket.eu.

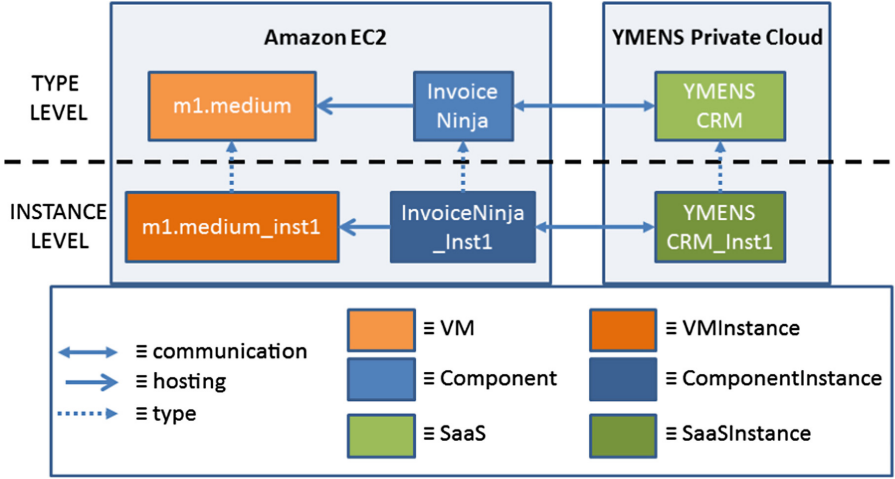


Fig. 1. The topology of the “SendInvoice” BPaaS.

or replace “YMENS CRM” with another SaaS. These rules are then given as input to the BPaaS Execution Environment of the CloudSocket platform which takes care of performing the respective adaptation actions required. The CEP-based SLO Evaluation framework proposed can be part of this environment by replacing an equivalent component which currently supports only adaptation at the IaaS level (mostly scaling actions).

$$R_1 : raw_cpu(m1.medium) > 80\% \wedge raw_mem(m1.medium) > 90\% \Rightarrow scale - out(IN)$$

$$R_2 : mean_rt(YC) > 20 \wedge mean_avail(YC) < 99.99\% \Rightarrow replace(YC)$$

$$R_3 : mean_cpu(m1.medium) > 70\% \wedge mean_rt(IN) > 20 \Rightarrow scale - out(IN)$$

where raw_cpu & raw_mem are the Raw CPU and Raw Memory Utilisation metrics, $mean_rt$, $mean_cpu$ and $mean_avail$ are the MEAN Response Time, CPU Utilisation and Availability metrics while IN represents the “Invoice Ninja” component and YC the “YMENS CRM” component.

Rules R_1 & R_3 focus on scaling out the “Invoice Ninja” component. The first rule attempts to immediately scale this component when one of its instances is severely overloaded. On the other hand, the third rule focuses on scaling out this component when its global status across all of its instances seems to be overloaded.

Rule R_2 attempts to replace the “YMENS CRM” external SaaS when both its mean response time is more than the threshold posed and its availability drops under a certain level. The replacement service is not specified as the system should dynamically find its replacement according to the current situation.

The whole specification of the use case in CAMEL, including the topology and the adaptation model of the “SendInvoice” BPaaS, can be found at: <https://drive.google.com/file/d/0B1oLQgQCVlqramYwa1hDZmtnSGc/view?usp=sharing>.

3 Related Work

Various approaches have been proposed in complex event processing and event pattern detection. Most rely on CEP engines that detect complex events continuously and build correlations and relationships between them, such as causality and timing ones. The detection of complex patterns is based on various techniques applied either over event streams [18, 19] or in an offline [9, 13] manner.

Statistical event detection approaches mainly exploit a user-defined minimum frequency or support (*minsup*). The springboard of all these approaches is the *Apriori* algorithm [1]. This algorithm produces the set of all significant association rules (rules relating a set of variables) between items in a large transactions database with a *minsup*. In [16], the authors introduce a method for discovering frequent event patterns, as well as their spatial and temporal properties in sensor networks, exploiting data mining techniques. Provided that events are put into a spatial and temporal context, the authors correlate certain event types on a sensor node with context events in a confined neighborhood in the recent past. Thus, a pattern of events is discovered whenever this pattern’s frequency surpasses a *minsup*. In [14], the authors propose the *Lossy Counting* widely used algorithm. This is an one-pass algorithm that computes approximate frequency counts of elements in a data stream and involves grouping the row items into blocks or chunks and counting within each chunk.

Temporal event processing approaches exploit the temporal relations among an input stream’s events. Such approaches can be very useful for deriving implicit information for the temporal ordering of raw data and predicting the future behavior of the monitored application. In [3] the authors introduce a formal framework for expressing data mining tasks involving time granularities, as well as algorithms for performing these tasks. Time constraints are injected into the system to bound the distance between an event pair in terms of time granularity. For instance, event e_2 must happen within two minutes after the occurrence of event e_1 so as to consider e_1, e_2 an event pattern. In [15] a temporal data mining approach is presented for data that cannot fit in memory or are processed at a faster rate than the generation one. The proposed sliding window model slides forward in hops of batches, while only a single batch is available for processing.

Moreover, logic-based approaches exploit inferencing to discover patterns defining respective association rules. In [17] a pattern discovery approach is proposed mapping logical equivalences based on propositional logic. In particular, a rule mining framework is introduced, generating coherent application domain independent rules for a given dataset that do not require setting an arbitrary *minsup*. The logic-based approach in [2] proposes an event calculus (EC) dialect, called *RTEC*, for efficient run-time recognition that is scalable to large data streams and exploits main EC predicates to discover specific activities. In our previous work [20], we have introduced a logic-based algorithm for discovering valid event patterns causing specific SLO violations. These event patterns interrelate events produced during the SBA’s execution and can be further exploited to enrich the adaptation rules defined by experts. This paper

goes a step further introducing a scalable and high-performance complex event processing framework that can realise and extend the event pattern detection feature.

Finally, other approaches also consider Business Process Management (BPM) when dealing with SLO evaluation. For instance, [6, 10] propose solutions (mainly scaling actions) for the optimization of Business Processes that are executed on virtualized environments. A similar approach is proposed in [8], where the authors apply data mining techniques to predict QoS and thus identify the correlation between the design and provisioning alternatives.

4 Background

4.1 Esper

Esper² is a stream-oriented CEP engine that provides the SQL-like and rich Event Processing Language (EPL). EPL enables expressing complex (event) matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, sorting and pattern detection. The proposed framework exploits it for the event pattern detection.

4.2 CAMEL

CAMEL is a multi-DSL, developed in the context of the PaaSage³ project to deal with the specification of multiple aspects in the multi-cloud applications lifecycle. It integrates already existing languages, like CloudML [7], as well of new languages developed with that project, like the Scalability Rule Language (SRL) [12]. The aspects covered by CAMEL mainly include: deployment, requirement, metric, scalability, provider and organisation aspects.

This paper focuses mainly on the metric and scalability aspects covered by the SRL sub-DSL of CAMEL. The metric package attempts to cover all measurement details that need to be specified for a non-functional metric, like formulas, functions, units of measurement plus measurement schedules and windows. This package is also able to specify conditions on metrics that can be exploited to specify SLOs as well as non-functional events in scalability rules.

The scalability aspect is covered via specifying scalability rules that map single events or event patterns to one or more scaling actions. Scaling actions can be either horizontal or vertical. Horizontal scaling actions include scale-out and scale-in actions while vertical actions include scale-up and scale-down.

The conceptualisation of events and event patterns is depicted in Fig. 2. Events can be single or composite. Single events can be further distinguished in functional and non-functional. Functional events map to a certain functional fault, like an application component failure. Non-functional events are associated

² <http://www.espertech.com/esper/>.

³ <https://paasage.ercim.eu/>.

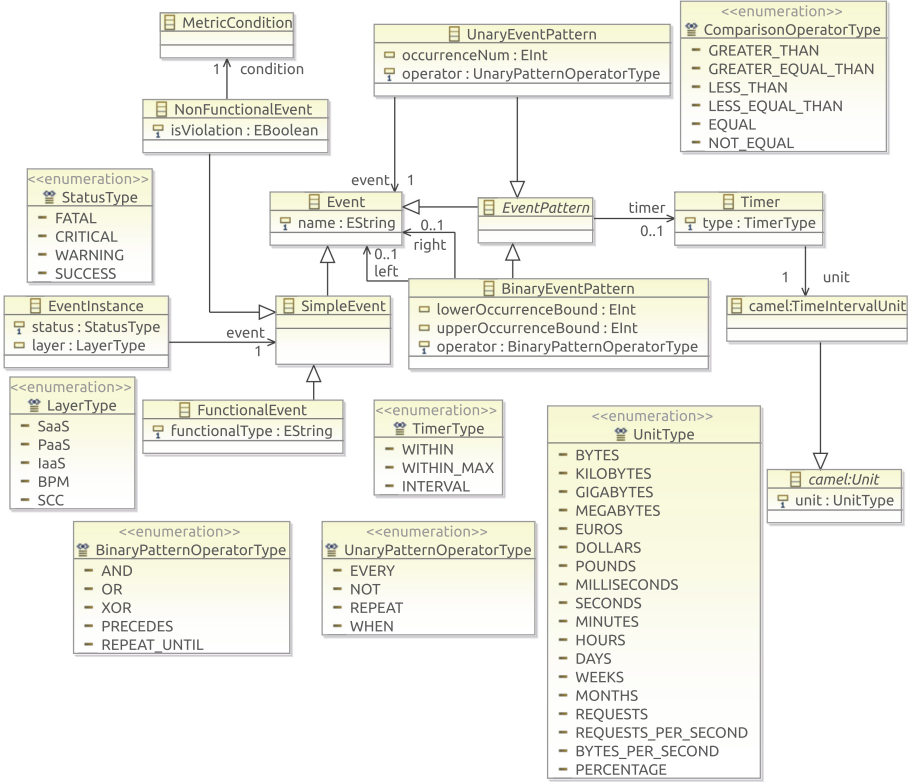


Fig. 2. The event pattern part of the SRL meta-model.

to a metric condition violation. A composite event maps to a logical or time-based combination of one or more events in the form of an event pattern. As such, such a combination is associated with respective logical and time-based operators. Both binary and unary operators can be defined which leads to producing unary and binary event patterns, respectively. Logical operators include AND, OR, NOT and XOR. Time-based operators have been inspired by Esper’s EPL and include many of the operators defined in that language (e.g., REPEAT).

As an event pattern is also a kind of event, patterns can be recursively defined. This means that, for example, when applying a binary logical operator (e.g., AND) over a certain binary event pattern, the first event could be single and the second could be another event pattern. For instance, suppose that the event pattern $EP_1: A \wedge (B \vee C)$ must be defined. To specify EP_1 , we need to define that the first event is A , the second event maps to the event pattern EP_2 and that the logical operator applied is \wedge . The second event pattern EP_2 would then be specified as the application of the \vee operator over two events, B and C .

As another example, consider the case of adaptation rules R_1 & R_3 which have the same consequent (i.e., adaptation action). In order to reduce the number

of rules that need to be checked and triggered by the system, these two rules could be combined into one. In that case and by considering that the name of each rule could also be the name of the respective event to be defined, then a more composite rule R_4 would be constructed which would map to the complex event pattern $(R_1 \vee R_3)$.

Via the recursive definition of events, more complex and advanced situations can be captured in respective rules. This should not stop to the case of scalability rules, but could cover any adaptation rule kind. This has been performed by the CloudSocket project (see footnote 1) [5] via an SRL extension. This extension can specify any adaptation rule kind at different abstraction levels. The event part of the rule specification was left as is, but the action part was extended to specify a workflow of adaptation actions that can be performed at the levels of infrastructure, platform, service and business process. As such, this extension fits well to the latest research trends in service computing that require specifying, executing and managing cross-layer rules to more effectively deal with the adaptation of cross-level SBAs in both simple and more advanced problematic situations.

In the context of this work, only the event part of an adaptation rule is considered due to intended functionality to be delivered. The CEP engine developed just detects the need to trigger a rule and then informs the rule execution component, e.g., an *Adaptation Engine*, to enact that execution of that rule.

5 SLO Evaluation Framework

5.1 Framework Analysis

The proposed SLO Evaluation Framework relies on the modular architecture depicted in Fig. 3. This architecture comprises three main levels: (a) interface; (b) core logic; (c) database (DB). At the interface level, the main actions (add, update, delete) that can be performed over an event pattern (EP) have been wrapped into the form of a REST service, called, *EP Service*, able to parse CAMEL/SRL fragments mapping to the specification of these patterns. Each action, when called, then has an impact over the core logic level of the framework.

At this second level, there is a main component, called *EP Parser*, which is responsible for processing the EPs obtained from the *EP Service*. Depending then on the action requested, different interactions take place at this level.

EP Addition. In case of adding a new EP, the *EP Parser* transforms it into an EP, specified in the EP language of the CEP framework, which is then registered in the server of that CEP framework, called *CEP Server*, so that it can be immediately detected. The names of metrics referenced by the EP, i.e., directly involved in the conditions of the EP's events, are also sent to the *Metric Subscriber* which not only informs its local metric list but also registers for subscribing to such metrics, when they are new, in the *Metric Publisher*. In parallel to this registration, the updated metric list of the *Metric Subscriber* is stored in the *EP DB* for fault-tolerance and rapid recovery reasons. The *Metric Publisher* is responsible

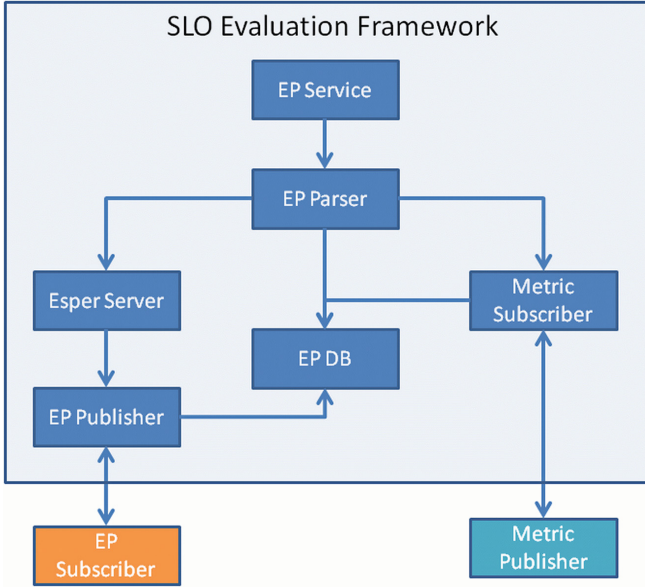


Fig. 3. The architecture of the SLO evaluation framework.

for publishing the values of metrics monitored to potential subscribers. As such, it can well map to a *Monitoring Engine* of a SBA management system. Once both the new EP and its respective metrics are registered in the corresponding system parts, the EP addition has been successful. So, the *EP Parser* stores the new EP in the *EP DB* not only for recovery reasons but also to gather statistics about EPs, while being detected by the *Esper Server*. The *EP DB* has been realised in the form of a model repository able to store, query and manipulate models of CAMEL, especially EPs, along with their statistics.

EP Deletion. In case of EP deletion, the EP is first fetched from the *EP DB*. Then, in parallel, the *EP Parser* informs both the *CEP Server* and the *Metric Subscriber* to update their structures and take further actions. The *CEP Server* just deregisters the EP's EPL specification. On the other hand, after checking that the EP metrics to be removed are not exploited in other EPs, the *Metric Subscriber* is informed to unsubscribe to these metrics to reduce the system load.

EP Update. In case of EP update, the produced EPL statement by the *EP Processor* is used to update the previous one. In addition, the *Metric Subscriber* is informed for adding or removing metrics which are or not needed any more (by any EP), respectively.

While the above actions can take place through the interaction of an external agent/user with the proposed Framework, we highlight that, in principle, the same interactions could be differently achieved, e.g., via a publish-subscribe

mechanism. As the respective functionality has been realised, we could easily switch from one to another mechanism or have both available at the same time.

As there are internally performed actions inside the framework, while it is running, these are now explicated in detail below.

As the *Metric Subscriber* subscribes to metrics, it can asynchronously receive measurements for such metrics from the *Metric Publisher*. Such measurements are then transformed into events which are fed into the *CEP Server*. Once all suitable events are received by the latter component, it can detect one or more EPs. When this occurs, this component will inform the *Event Publisher*.

The *EP Publisher* is responsible for publishing events to interested subscribers, named as *EP Subscribers*. Such subscribers could be adaptation engines responsible for executing the respective adaptation rule triggered, as, e.g., specified in CAMEL. Apart from this publication, the *EP Publisher* also updates the entries in the *EP DB* to modify the respective statistics of the EP(s) concerned.

The proposed architecture exploits publish-subscribe mechanisms to both receive some events/measurements and publish other kinds of events (e.g., EPs). In this respect, it can actually interact with multiple components that might be willing to obtain information from or feed information to this framework. For instance, adaptation responsibility for an SBA management system could be split into multiple instances of an *Adaptation Engine* to balance the respective load. All these instances could then subscribe to the *EP Publisher* to manage their own part of the adaptation space, i.e., only those EPs that concern them.

The presented architecture is logical. This means that it can be flexibly distributed at the physical level. For instance, we could have multiple instances of the framework part that involves the *CEP Server* and the *Event Publisher* to load balance the event workload entering the framework. Alternatively, we could scale out the whole framework into parts that focus on different EP partitions. For example, the SBA management system could be split similarly into different parts, where each part could be devoted to a subset of all SBAs managed. Each system part could be then associated to one instance of the SLO Evaluation Framework, thus mapping only to the EPs of the SBAs that need to be handled.

5.2 Implementation

All framework components have been implemented in Java. The CEP engine exploited is Esper, version 5.3.0. For the publish-subscribe mechanism, the 0-MQ⁴ messaging middleware has been exploited that incurs less overhead with respect to other messaging middleware realisations. The *EP DB* has been realised as a model repository implemented via the CDO technology⁵ which provides suitable and robust mechanisms for model persistence and lazy loading as well as the HQL language to enable posing queries at a higher abstraction level than pure SQL. The *EP Service* has been implemented via the Jersey⁶ java library.

⁴ zeromq.org.

⁵ <https://eclipse.org/cdo/>.

⁶ <http://jersey.github.io/>.

6 Event Pattern Generation and Detection

While it could be considered as straightforward to transform an event pattern in CAMEL into an EPL statement in Esper, this is by far not trivial as the events in an EP need to be correctly correlated. Correlation means that the events should be associated with either the same measured components or with components that are connected in the SBA dependency hierarchy. This also has an impact on the way measurements are represented as the information concerning the measured component should be already present and be then copied accordingly in the internal representation of the event in Esper.

Concerning the metric measurements, we have actually assumed the following: (a) the *Metric Subscriber* subscribes only to metrics based on their name; (b) the *Metric Publisher* publishes measurements for metrics that might be named equivalently. The latter means that the measurement information published should include sufficient information to enable the framework to identify exactly what object is being measured.

To decouple the proposed framework from the dependency knowledge it should possess, we assume that such dependency information is provided within the measurement information published. While this leads to some published information duplication, it translates to a loose integration of this framework with the SBA management system. Otherwise, the framework would need to connect to a `models@runtime` component [4] in that system to be informed constantly about both the type and instance level in the SBA dependency hierarchy.

The measurement information published includes: (f1) the metric’s name (e.g., *MeanResponseTime*); (f2) the metric value; (f3) the measurement timestamp; (f4) the name of the application/service concerned; (f5) the name of the component measured; (f6) the name of the instance of the component measured; (f7) the name of the VM measured; (f8) the name of the instance of the VM measured.

Values for fields f1–f4 are always present. Depending on the level and kind of component measured, only some of the values of the other fields need to be supplied based on the following cases mapping to the type of measurement:

- *ApplicationMeasurement*: here the measurement concerns the whole application so no additional fields are needed.
- *VMMeasurement*: here the measurement concerns a certain VM. There are two sub-cases holding now: (i) the measurement concerns the VM type (e.g., `m1.medium`) and not its instance. Then, only the field f7 has to be provided; (ii) the measurement concerns the VM instance (e.g., `m1.medium_inst1`). In this case, we need to provide both fields f7 & f8 as the type of the VM instance concerned needs to be provided.
- *ComponentMeasurement*: here the measurement concerns a certain (software) component. Again, two sub-cases might hold: (i) the measurement concerns the component type (e.g., “InvoiceNinja”). In this case, apart from field f5, we also need to provide field f7 (thus provide, e.g., the value of “`m1.medium`”) as one component might logically be deployed into multiple VMs within the

same deployment topology. As such, we need to explain for which deployment the current measurement holds; (ii) the measurement concerns a component instance (e.g., “InvoiceNinja_Inst1”). In this case, all the fields need to be provided in order to cover both the deployment of that component instance at the instance level as well as the deployment of its (component) type at the type level. Thus, considering the running example/use case, the following values for the measurement fields will be provided: f5=“InvoiceNinja”, f6=“InvoiceNinja_Inst1”, f7=“m1.medium”, f8=“m1.medium_inst1”.

By explaining how measurements are structured and instantiated based on the kind of the component concerned, now we will explain the way EPs in CAMEL are transformed into EPL statements. We distinguish between two cases – (C_1) all events in the EP refer to the instance level; (C_2) all events in the EP refer to the type level. We do not consider a mixture of events from different levels as this does not make sense.

The C_1 case maps to two sub-cases:

1. all events refer to the same component. Consider, for instance, the case of rule R_1 . The respective EPL statement to be created for this rule would be the following:

```
every(ev1=Event(metric='CPUUtilisation' and value >= 80
and application='SendInvoice' and vm='m1.medium') and
Event(metric='MemoryUtilisation' and value >= 90 and appli-
cation='SendInvoice' and vmInstance=ev1.vmInstance and
vm='m1.medium'))
```

In this statement, we join these two events as streams based on their application, VM and VM instance fields. Via this join, we impose that the EP should hold for a specific application and VM but we do not care about which matched vm instance is concerned (as any instance needs to be matched here). Moreover, the presence of EVERY indicates that the pattern should be repeatedly inspected and not just once.

2. all events refer to different but related components. In this case, the EPL statement under construction needs to correlated the different components together. For instance, suppose that an alternative rule to R_1 would attempt to scale the “InvoiceNinja” component when its raw CPU utilisation is above 80% and its response time is above 20 s. The respective statement generated for this alternative rule would take the following form:

```
every(ev1=Event(metric='CPUUtilisation' and value >= 80
and application='SendInvoice' and vm='m1.medium') and
Event(metric='ResponseTime' and value > 20 and vmIn-
stance=ev1.vmInstance and application='SendInvoice' and
vm='m1.medium' and component='InvoiceNinja'))
```


This EPL statement is more complicated as it needs to join two events for which we need to guarantee that they refer to the same application, VM and VM instance, where the first two fields are mapped to specific values. We also need to guarantee that the second event refers to the “InvoiceNinja” component but we do not care about the instance of that component as we guarantee that the same VM instance, as in the first event, has been used to deploy this particular instance of that component.

For the type level, we have the following two similar kinds of cases which, however, lead to the construction of simpler EPL statements.

1. all events refer to the same component. For instance, suppose that Rule R_2 applies here. Then, the respective EPL statement to be constructed would take the following form:

```
every(ev1=Event(metric='MeanResponseTime' and value > 20 and
application='SendInvoice' and component='YMENS CRM') and
Event(metric='MeanAvailability' and value < 99.99 and applica-
tion='SendInvoice' and component='YMENS CRM'))
```

In this statement we just join two event streams based on their application and component which are clearly identified in the respective conditions.

2. the events refer to different but correlated components. For instance, suppose that Rule R_3 needs to be applied. The respective EPL statement to be constructed will be the following:

```
every(ev1=Event(metric='MeanResponseTime' and value > 20
and application='SendInvoice' and component='InvoiceNinja' and
vm='m1.medium') and Event(metric='MeanCPUUtilisation' and value
> 70 and application='SendInvoice' and vm='m1.medium'))
```

So, we actually join again the two events by considering the following: (a) the join is made based on the application and VM fields; (b) for the first event, we need to identify the correct component concerned; (c) for the second event, we do not need to specify a respective software component as it concerns the infrastructure (VM) level.

Cases C_1 and C_2 with their 2 sub-cases have been exemplified via certain examples. In reality, our framework is able to go beyond the capabilities shown in these examples. It can process any kind of complex EP with an arbitrary nesting and any kind of operator from those captured by CAMEL. However, showing such a complex case needs substantial space and thus, it has been left out from the analysis in this paper.

7 Conclusions and Future Work

This paper has proposed a new SLO evaluation framework for SBAs that relies on a rich EP expression language, namely SRL, and on the well-known Esper CEP engine. This system has been designed based on a modular architecture where many of its parts can scale on demand. This system is also loosely coupled with the respective monitoring and adaptation engines that might be employed in a SBA management system. The management of EPs is wrapped into the form of a REST service enabling a respective SBA management system to be decoupled from underlying implementation peculiarities and manage the generation and handling of adaptation rules that contain such EPs.

Concerning future work, we plan to further evaluate the SLO evaluation framework and especially investigate its exact distribution points. We also plan to compare Esper with other CEP engines in order to reach an informed decision about which CEP engine is more suitable in our context. In fact, it can be interesting to create a system which can be configured to exploit different CEP engines by incorporating the appropriate abstraction mechanisms.

Acknowledgments. This work is supported by CloudSocket project that has been funded within the European Commission’s H2020 Program under contract number 644690.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)
2. Artikis, A., Sergot, M.J., Paliouras, G.: Run-time composite event recognition. In: DEBS, pp. 69–80. ACM (2012)
3. Bettini, C., Wang, X.S., Jajodia, S., Lin, J.-L.: Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Trans. Knowl. Data Eng.* **10**(2), 222–237 (1998)
4. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. *Computer* **42**(10), 22–27 (2009)
5. Seybold, D., Griesinger, F., Kritikos, K., Gallo, A., Cacciato, S., Popovici, A., Iranzo, J., Sosa, R., Utz, W., Falcioni, D.: Explanatory Notes: Final BPaaS Prototype. CloudSocket Project Deliverable D4.6–D4.8, June 2017
6. Euting, S., Janiesch, C., Fischer, R., Tai, S., Weber, I.: Scalable business process execution in the cloud. In: 2nd IEEE Conference on Cloud Engineering (IC2E), pp. 175–184. IEEE (2014)
7. Ferry, N., Chauvel, F., Rossini, A., Morin, B., Solberg, A.: Managing multi-cloud systems with CloudMF. In: NordiCloud, pp. 38–45. ACM (2013)
8. Ghosh, R., Ghose, A., Hegde, A., Mukherjee, T., Mos, A.: QoS-driven management of business process variants in cloud based execution environments. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 55–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_4
9. Hellerstein, J.L., Ma, S., Perng, C.-S.: Discovering actionable patterns in event data. *IBM Syst. J.* **41**(3), 475–493 (2002)

10. Janiesch, C., Weber, I., Menzel, M., Kuhlenkamp, J.: Optimizing the performance of automated business processes executed on virtualized infrastructure. In: 47th Hawaii International Conference on System Sciences (HICSS), pp. 3818–3826. IEEE (2014)
11. Kazhamiakin, R., Pistore, M., Zengin, A.: Cross-layer adaptation and monitoring of service-based applications. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave - 2009. LNCS, vol. 6275, pp. 325–334. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16132-2_31
12. Kritikos, K., Domaschka, J., Rossini, A.: SRL: a scalability rule language for multi-cloud environments. In: CloudCom. IEEE (2014)
13. Magnusson, M.S.: Discovering hidden time patterns in behavior: T-patterns and their detection. *Behav. Res. Methods Instr. Comput.* **32**(1), 93–110 (2000)
14. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams, pp. 346–357 (2002)
15. Patnaik, D., Ramakrishnan, N., Laxman, S., Chandramouli, B.: Streaming algorithms for pattern discovery over dynamically changing event sequences. *CoRR*, abs/1205.4477 (2012)
16. Römer, K.: Distributed mining of spatio-temporal event patterns in sensor networks. In: EAWMS Workshop at DCOSS, pp. 103–116 (2006)
17. Sim, A.T.H., Indrawan, M., Zutshi, S., Srinivasan, B.: Logic-based pattern discovery. *IEEE Trans. Knowl. Data Eng.* **22**(6), 798–811 (2010)
18. Wang, D., Rundensteiner, E.A., Ellison, R.T.: Active complex event processing over event streams. *PVLDB* **4**(10), 634–645 (2011)
19. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD Conference, pp. 407–418. ACM (2006)
20. Zeginis, C., Kritikos, K., Plexousakis, D.: Event pattern discovery for cross-layer adaptation of multi-cloud applications. In: Villari, M., Zimmermann, W., Lau, K.-K. (eds.) ESOC 2014. LNCS, vol. 8745, pp. 138–147. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44879-3_10
21. Zeginis, C., Kritikos, K., Plexousakis, D.: Event pattern discovery in multi-cloud service-based applications. *IJSSOE* **5**(4), 78–103 (2015)



Towards Business-to-IT Alignment in the Cloud

Kyriakos Kritikos¹(✉), Emanuele Laurenzi², and Knut Hinkelmann²

¹ ICS-FORTH, Heraklion, Greece

`kritikos@ics.forth.gr`

² FHNW University of Applied Sciences and Arts Northwestern Switzerland,
Olten, Switzerland

`{emanuele.laurenzi,knut.hinkelmann}@fhnw.ch`

Abstract. Cloud computing offers a great opportunity for business process (BP) flexibility, adaptability and reduced costs. This leads to realising the notion of business process as a service (BPaaS), i.e., BPs offered on-demand in the cloud. This paper introduces a novel architecture focusing on BPaaS design that includes the integration of existing state-of-the-art components as well as new ones which take the form of a business and a syntactic matchmaker. The end result is an environment enabling to transform domain-specific BPs into executable workflows which can then be made deployable in the cloud so as to become real BPaaSes.

Keywords: BPaaS · Service · Design · Discovery · Selection
Alignment · Mediation

1 Introduction

Due to intense market competition, organisations can survive only if they offer services that are either innovative or exhibit a better quality than their competitors. However, by owning a limited infrastructure and continuously requiring to improve the existing business processes (BPs) leads to reaching certain impassable limits. Moreover, the infrastructure maintenance, operation and management costs can be quite prohibiting, especially for small or medium enterprises.

Fortunately, cloud computing can become the medium via which organisations can acquire cheap, commodity resources on-demand while also being able to achieve certain benefits, including: outsourcing infrastructure management with reduced cost, flexible resource management, and elasticity. Such benefits can certainly enable improving and optimally controlling BP performance.

However, as cloud computing handles only the infrastructure level, an organisation now faces the hard and yet unsolvable problem of aligning the business with the IT level. Moreover, many organisations do not have the expertise and know-how to use and combine the cloud services offered.

The above problems can be solved by combining BP management with cloud computing to realise the BP as a service (BPaaS) paradigm to enable migrating and more optimally managing BPs in the Cloud [6, 31, 32]. However, such a

combination is not trivial as it leads to the following challenges which especially concern the BP design lifecycle activity: (a) how to map a BP to a technical workflow with a suitable automation level; (b) how to align business terms and requirements with technical ones to drive the selection of the most suitable services to be then integrated into the workflow; (c) how to deal with the service incompatibility problem effectively to guarantee the correct execution of the designed workflow. Such a problem relates to checking the syntactic compatibility of messages exchanged between two or more selected workflow services.

To realise the vision of BPaaS, the CloudSocket project (www.cloudsocket.eu) delivers a platform that unifies together environments supporting different BP lifecycle activities. This paper presents our contribution in form of a BPaaS Design Environment able to deal successfully with all aforementioned challenges. This translates to introducing an innovative architecture with suitable components that support: smart and semantic service discovery at both business and technical levels, optimal cross-level service selection, mapping between business and technical requirements and mediation between the execution of two or more services to achieve message-level compatibility. In result, the developed environment enables a BPaaS provider to transform the initial business functional and non-functional requirements that match the necessities of potential BPaaS customers into an executable workflow. That workflow can then become deployable in the cloud by using other CloudSocket environments.

The BPaaS Design Environment was built by exploiting state-of-the-art as well as two novel components. The first component, the business matchmaker, enables to find services that satisfy the user functional and non-functional requirements at the business level by following a novel questionnaire-based approach. Such services are then filtered and selected by employing state-of-the-art technical service matchmaking and selection components. Service selection relies on the second novel component, the syntactic matchmaking one, able to infer the message-based compatibility between two or more selected services and produce a mapping specification. This specification can then be exploited by a service mediation service to support the compatible message transformation between services and thus guarantee the smooth operation of the BPaaS workflow in which this mediation service is integrated.

This paper is structured as follows. Section 2 shortly analyses existing research results, some of which are exploited in the production of the BPaaS Design Environment. Section 3 analyses the environment's main architecture by also explaining the main functionality and role of its components. Sections 4 and 5 detail the architecture's two main novel components, the business and syntactic matchmakers. Section 6 introduces a use case to demonstrate the main benefits of the proposed environment and to validate it. Finally, the last section concludes the paper and draws directions for further research.

2 Background

2.1 Business-to-IT Alignment

Business-to-IT alignment typically refers to the gap between business requirements and technical solutions [12]. Cloud offerings are technically described making it hard for business people to properly assess the best fitting cloud solution [33]. Thus, identifying suitable cloud solutions requires specifying requirements for and capabilities of a service in both a business and IT language. To ensure knowledge understandability and transparency, it is a common practice to represent knowledge in models [11, 29]. Models abstract away from complex realities and achieve precise modelling of the intended domain. In [13] we already adopted a model-driven approach where an extension of BPMN 2.0 allows modeling both BP requirements for business and workflows/cloud services in a technical language. That approach includes translating the business to the technical language to enable matching process requirements and workflow/cloud service capabilities. Translation and matching are performed by semantically lifting models with ontologies to make them machine-interpretable.

[2] defines *Semantic Lifting* as “the process of associating content items with suitable semantic objects as metadata to turn unstructured content items into semantic knowledge resources”. Semantic Lifting shifts the purpose of modelling beyond transparency and communication [14]. The interpretable knowledge base (ontology) allows reaching higher system automation levels based on models [13]. For example, an ontology-based early warning system assessing supply chain risks was proposed in [8], while in [7] ontologies are combined with a case-based reasoning approach to support workplace learning. Closer to our current problem, [9] introduced the AML ontology for automatic identification of correspondences between BP model activities. Similar BP matching approaches are described in [1]. Such approaches are not sufficient for BP-to-workflow matching as a BP is far less detailed than a workflow such that a BP activity is most likely to refer to a whole workflow fragment. As such, due to this different degree of detail between the two levels, such approaches suffer from inaccurate matching, something not only addressed but also far improved by our approach.

Approach. We follow a model-driven which performs domain-specific conceptualization (mapping to well-known benefits [10, 17, 25]) on two levels, where the one targets BP users, while the other targets IT service experts. This allows designing domain-specific models capturing suitable domain knowledge on both levels. This approach builds upon the findings in [13] but adopts a different perspective on business-to-IT alignment in the Cloud. Namely, there is a shift from language translation to the mapping of values between requirements and specifications on both the business and IT levels, separately. Hence, the Business-IT alignment paradigm is applied sequentially by further refining results from the business to the IT level. As such, 3 matchmaking components are proposed: (a) the business and (b) technical matchmakers enhanced with formal semantics for machine-interpretation plus (c) the syntactic matchmaker. The combination of these 3 components allows identifying the most suitable cloud services within both business and technical terms that will eventually form a workflow.

2.2 Technical Service Matchmaking

Technical service matchmaking involves functional and QoS matching. Functional matching usually focuses on I/O-based matching [18, 26] while QoS matching takes the view of QoS as conformance [20] and employs different kinds of techniques [21] to infer if the service's solution space is included in that of the request. While most work focuses on one aspect individually, some approaches consider both aspects simultaneously [3, 16]. However, they usually sequentially combine the matching in both aspects and do not employ semantic techniques, thus not exhibiting the right performance and accuracy level.

As such, our previous work [23] explored different ways the 2 matching types can be jointly performed: (a) sequential combination; (b) parallel combination; (c) subsumes-based combination. The experimental evaluation of these combinations showed that the parallel one leads to the best possible results with respect to performance, as matchmaking accuracy is perfect in all combinations.

Our approach exploits two aspect-specific matchmakers, a functional and a non-functional. The functional is a state-of-the-art matchmaker developed in the Alive project [4] which relies on the combination of I/O-based and IR-based matching. It exploits a smart graph-based structure to dynamically tolerate changes in domain ontologies (i.e., the ontologies via service I/O is annotated) as well as supply almost constant-in-time query operations over the graph.

The unary matchmaker [21] follows a hybrid QoS service matching approach. First, it aligns ontology-based service specifications based on their QoS terms. Then, it performs service filtering in a step-wise manner by considering each QoS term individually in each step. As unary constraints are assumed to be involved in service offers and demands, the matchmaker employs smart structures to support term-based filtering which results in ultra fast matching time.

2.3 Service Selection

Service selection work usually considers only one abstraction level by also neglecting semantics, thus producing results of imperfect accuracy. Accuracy is further reduced as some algorithms employ smart but non-optimal solving techniques, like Genetic Algorithms to accelerate the service selection time.

As service selection for a BPaaS includes different abstraction levels, we have developed a cross-level constraint-based algorithm [22] which exhibits the following features: (a) handling of multiple optimisation objectives by employing the Analytic Hierarchy Process (AHP) [27] and Simple Additive Weighting (SAW) [15] techniques; (b) the capability to bridge the gap between the two levels (SaaS and IaaS) via inserting functions that derive the QoS at the SaaS level based on the capabilities selected at the IaaS level; (c) the addressing of overconstrained requirements by employing smart utility functions that allow slightly violating these requirements so as to produce at least one solution; (d) consideration of dependencies between QoS parameters at the same level enabling a more accurate evaluation of respective solutions; (e) the capability [19] to accelerate solving time by fixing parts of the problem to certain partial solutions by relying on the BPaaS execution history.

State-of-the-Art Advancement. The proposed BPaaS Design Environment advances the state-of-the-art by exhibiting an innovative combination of existing, like holistic technical service matchmaking, and new features. The innovative business matchmaker follows a dynamic questionnaire-based approach enabling business users to answer a minimum set of questions before the mapping of the designed BP to a set of services, able to realise its functionality, can be produced. Such an approach is more natural and user-intuitive as it employs questions mapped to a natural language with terms drawn from the business domain. It also supports producing a minimal set of services to be further filtered and selected based on technical requirements such that the solution space is significantly reduced and service discovery time accelerated.

The novel syntactic matchmaker enables producing a correct executable workflow via the suitable integration of services at the technical level based on their message compatibility. Such compatibility is guaranteed by generating mapping specifications that are exploited by mediation tasks incorporated in the generated workflow. Finally, our framework addresses all layers involved in a BPaaS system along with their dependencies thus being able to produce a more complete and optimal BPaaS design product.

3 Architecture

The creation of the BPaaS Design Environment was underpinned by the design science research (DSR) methodology in [30]. First, the literature on Business-IT alignment in the Cloud was screened. Then, CloudSocket created the settings to contribute to the problem awareness: application scenarios were created in workshops involving both industrial and scientific experts. The results and insights were useful to suggest the BPaaS Design Environment's first draft which was then finalised in a web-based solution through continuous development. Finally, as shown in Sect. 6, the validation took place with respect to the most agreed application scenario among the members of the CloudSocket consortium.

The BPaaS Design Environment follows a model-driven and semantics-aware approach for business-to-IT alignment in the cloud which comprises 3 main transformation steps: (a) BP-to-business-services; (b) business-services-to-technical-services; (c) BP & technical-services-to-executable-workflow. The approach guarantees the produced solution's technical feasibility by employing a two-step service matchmaking process at both business and technical levels and a service selection algorithm that is syntactic-compatibility-aware at the technical level.

To achieve its main goal, the environment exhibits an architecture, depicted in Fig. 1, comprising 8 main components that are now analysed in detail. Some components correspond directly to some of the aforementioned steps while others play a supporting or orchestration role.

BPaaS Designer (BD). It represents the main point of interaction with the user during BPaaS design. It enables specifying both domain-specific BPs and executable workflows. It also guides users in providing suitable input to support the BP-to-workflow alignment.

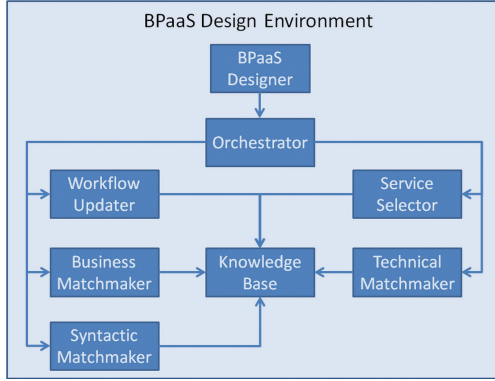


Fig. 1. The architecture of the BPaaS Design Environment

Orchestrator (Orch). Orchestrates its underlying components to handle requests issued by the BPaaS Designer.

Business Matchmaker (BM). Matches the cloud services registered in the Knowledge Base based on business requirements derived from a questionnaire-based approach explained in Sect. 4.

Technical Matchmaker (TM). It exploits technical state-of-the-art aspect-specific matchmakers in a parallelised fashion according to the approach in [23].

Service Selector (SS). It [22] produces a concrete optimal solution for the service-based workflow at hand by considering the user technical non-functional requirements while also attempting to maximise the message compatibility between services by exploiting the next component.

Syntactic Matchmaker (SM). Called dynamically by the SS while solving the service selection problem to find the message compatibility [24] between the next and all previously selected services in each BPaaS workflow’s execution path where such a service participates. When an incompatible solution is constructed, SS can backtrack and check another one. To smartly deal with cases where the same call is issued, e.g., due to deep backtracking, SM stores the call results to immediately answer it. The mapping of the output parameters to the input ones of the next service is also recorded to enable updating the BPaaS workflow via a mediation service, as performed by the next component.

Workflow Updater (WU). Updates the BPaaS workflow by performing the following actions for each workflow’s execution path: (a) replays the solution construction in each path to obtain the mapping of the current service in the path from the SM; (b) introduces a mediation service within the workflow, immediately before the current service, which takes as input the current output parameter set and the mapping specification and produces as output the input parameters of the current service.

Knowledge Base (KB). Includes all necessary and sufficient information to support all reasoning/matching/selection tasks executed in the system.

4 Business Matchmaking

The *Business Matchmaker* allows specifying requirements in a more user-centric approach than that in [13]. It relies on a context-adaptive questionnaire that guides the user via a set of questions reflecting BP functional and non-functional requirements. Follow-up questions are displayed based on the result of a prioritisation algorithm that considers: (a) user preferences in terms of categories (e.g., *Performance* rather than *Data Security*); (b) information value (or entropy) of semantic attributes reflecting cloud service specifications at the business level, e.g., how distinguishing an attribute, such as *monthly downtime*, is for service filtering. Namely, the higher the entropy value of an attribute, the higher its service distinguishability degree, and thus the higher the assigned priority of the related question. This approach leads to the least possible number of questions being answered, thus reducing the business service matching time. The idea is that the questionnaire can be applied on the whole BP first. If no service is found, we then move down to groups of activities, until the level of single activities.

4.1 The Context-Adaptive Questionnaire

The Context-Adaptive Questionnaire relies on our BPaaS ontology [11]. Questions focus first on functional requirements and then on non-functional ones. The questionnaire enables the user to specify functional requirements in two ways by:

- inserting an action and object from a predefined taxonomy in the BPaaS ontology. This corresponds to the convention of BPMN to name activities by a verb (i.e., action) and a noun (object) [28] whose combination provides the “what-is-about” knowledge.
- inserting the most suitable category from APQC Process Classification Framework.

Next, the user can choose one of the 5 non-functional (NF) categories: *Data Security*, *Payment*, *Performance*, *Service support*, and *Target Market*.

The NF categories were derived from the Cloud Service Agreement Standardisation Guidelines [5], published by EC to standardize and streamline the terminologies and understanding of cloud services. The NF categories were subsequently discussed and validated within the CloudSocket consortium. In result, a set of questions and sub-questions were derived out of them. For instance, the *Performance* category includes questions like the following:

- What is your preferred monthly downtime in minutes?
Possible answer: 30 min
- Should the process be executed on a daily, weekly, monthly or yearly basis?
Possible answer: On a weekly basis

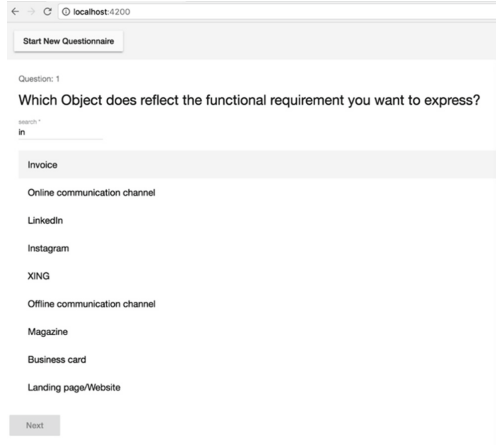


Fig. 2. The object selection for the functional requirements posing

- What is your favorite response time level?
Possible answer: High, Medium or Low
- How many simultaneous users should the cloud service support?
Possible answer: at most 10

For each question, we have distinguished among 4 types of answers as: (1) single-answer selection; (2) multi-answer selection; (3) search-insert; (4) value-insert. Value- and search-insert require user input. While the former enables inserting attribute values (e.g., the aforementioned downtime), the latter enables crawling predefined values from the ontology and selecting the suitable one. For instance, answers related to the first 3 functional requirement questions (Action, Object and APQC category) are of search-insert type. Namely, users can insert keywords for the BP they are looking for, and the ontology returns the concepts matching these keywords. Figure 2 shows this functionality’s implementation result.

Each time a question is answered, semantic rules are applied to convert implicit knowledge reflecting the business requirements into an explicit one. This prepares the ground to identify matching cloud services by applying a semantic query. For example, assume we have the following:

Specifications from the KB as follows:

- A cloud service with the execution constraint of 20 times per day.

Requirements from the questionnaire as follows:

- Should the process be executed on a daily, weekly, monthly or yearly basis?
Answer: At least on a weekly basis.
 - How many times should the process be executed?
Answer: At least 10 times

Running a process at least on a weekly basis implies that can also run on a daily basis. The semantic rule, therefore, would infer the answer “On a daily basis” and insert it in the KB. The semantic query then compares the derived fact with the cloud service fact related to the execution constraint. In result, the cloud service specification matches with the requirement.

4.2 Question Prioritisation Algorithm

The NFR questions follow a question prioritisation algorithm. This enables identifying the matching cloud services by asking as few questions as possible. Answers to the questions, along with previous ones, are used to display the follow-up question. The algorithm considers the following:

- Grouping among non-functional attributes. For instance, if the user selects to answer one from *availability* and *response time* attributes of the *Performance* category, the follow-up question will be on the other attribute in this category.
- Entropy expressing the variation degree in the values of each non-functional attribute. Entropy of an attribute is “0” when every cloud service stored in the *KB* contains the same attribute value, while “1” in the opposite case.

The entropy formula is expressed as follows:

$$Entropy(attr_i) = - \sum_{j=1}^J (p_{ij} \cdot \log_2(p_{ij}))$$

where J is the total number of attribute values and p_{ij} is the probability that a certain attribute value val_{ij} of attribute $attr_i$ appears in a certain cloud service. As this probability can be regarded as independent and uniform across all attribute values, p_{ij} can be expressed as: $p_{ij} = \frac{[CS]_{cval_{ik}=val_{ij}}}{[CS]}$ where the nominator denotes the number of cloud services that exhibit the respective attribute value ($cval_{ik}$ denotes the value of $attr_i$ for cloud service k) and the denominator the number of all services.

The prioritisation algorithm’s signature and main logic is as follows.

Input.

- Already stated variables: $attr, CS, val, cval$.
- The set of non-functional categories $C = \{Data\ Security, Payment, Performance, Service\ support, Target\ Market\}$.
- Set of tuples $\langle attr_i, Q_l \rangle$ where Q is the set of questions and Q_l is a certain question where $1 \leq l \leq [Q]$. So, each tuple maps 1 attribute to 1 question.

Output. The filtered set of cloud services CS that match with the content of the questionnaire, i.e., questions and answers.

Business Logic.

1. IF the number of categories left is positive ($|C| > 0$), select a category c_n , ELSE exit.

2. IF c_n has a positive number of semantic attributes left, i.e., $|attr_i \text{ s.t } attr_i.cat = c_n| > 0$, THEN calculate the entropy of all the selected category's attributes, ELSE remove the current category c_n from C and go to (1).
3. Select attribute $attr_i$ with highest entropy.
4. Display question Q_l that is mapped with the $attr_i$.
5. Get user answer mapping to a value val_{ij} of attribute $attr_i$.
6. Filter services in CS which do not satisfy the condition: $csval_{ik} = val_{ij}$.
7. Remove the semantic attribute $attr_i$ from the category c_n and go to (2).
8. Exit.

5 Syntactic Matchmaking

Business/technical matching cannot guarantee the message compatibility between selected services in a BPaaS workflow. Such a compatibility is thus a hard constraint in service selection for producing optimal, message-compatible solutions that can be safely executed. As such, the TM was developed to derive such compatibility and offer it as a function to SS.

The main idea is that the TM should first find which output messages of previously selected services match to which input messages of the currently selected service (based on SS's solution generation process) for each execution path in the BPaaS workflow. Then, it should check for each message-to-message match if the first message conveys less information than that required by the second message. If this checking succeeds, no compatibility between the execution path's considered services exists. When all message pair matches are compatible, the considered services are message-compatible.

Message Matching. The first message compatibility step can rely on existing semantic service annotations to easily and rapidly discover matching message pairs, as the messages involved in these pairs should map to semantically compatible concepts. However, even in the presence of such knowledge, message matching is not trivial and follows a two-step process involving semantic & syntactic message matching. This process is exemplified via the example of a certain service pair involving service S_2 with 2 input parameters mapped to ontology concepts A & B and service S_1 with 2 output parameters mapped to ontology concepts C & D .

At the semantic level, a bipartite matching approach is followed checking whether every parameter of the current service has a mapping to one parameter of the previously selected services (or the initial user input) in a certain execution path and attempting to discover a solution with the lowest overall distance. As such, we first define a local matching degree between two parameters to be the distance between the parameters' annotation concepts in the ontology subsumption hierarchy, provided that the second parameter's concept subsumes the first parameter's one. If the latter does not hold, the distance is infinite. This guarantees that no information loss occurs as in the opposite case, the more concrete concept in the S_2 input will require specifying additional pieces

of information than those exhibited in the concept in the S_1 output. A mapping solution's overall distance is then the sum of the distances of the matches found. As such, the matching problem can be defined as follows:

$$\min \left\{ \begin{array}{l} \frac{1}{|J|} \cdot \left(\sum_{i \in I} \sum_{j \in J} \left(\frac{\text{dist}(M_i, N_j)}{\text{maxPSize}} \cdot x_{ij} \right) + \sum_{j \in J} (1 - \sum_{i \in I} x_{ij}) \right) \\ \sum_{j \in J} x_{ij} \leq 1 \\ \sum_{i \in I} x_{ij} \leq 1 \\ i = [1, \dots, [I]], j = [1, \dots, [J]] \end{array} \right\}$$

where I and J are the sets of input and output parameters, respectively, x_{ij} is a decision variable whether the output parameter i matches the input one j , $\text{dist}(M_i, N_j)$ is the distance between annotation concepts M_i and N_j of the two parameters pair while maxPSize represents the maximum subsumption path length in the respective domain ontology used.

Suppose that the following relations hold in the running example: A subsumes B , C & B , C subsume D . In this respect, the best possible matching is $\{A \rightarrow C, B \rightarrow D\}$ with overall distance of 2. The other matching solution $\{A \rightarrow D, B \rightarrow C\}$ is not selected as the local distance between B & C is infinite so the overall distance is also infinite.

The algorithm then proceeds at the syntactic level by considering only those message pairs with a finite local degree of match. For each message pair filtered, we note the information items for the output parameter and those of the input parameter and then we check whether the former include the latter. As the information items have been already matched to ontology concepts, we perform this checking by replacing the information items with the attributes of the ontology concept. Even if the concepts matched are not identical, as they are related with a subsumption relation, they will have common attributes. So, the problem then is mapped to checking whether the concept attributes of the output parameter form a superset of those of the input parameter.

Message types might also convey information not included in an ontology requiring to perform a different matching kind for them. This matching's logic is similar to that for the semantic level. In particular, bipartite matching is performed with the exception of how the distance is calculated at the local level. At that level, we consider both how similar the field names are and how close are their types. Name similarity can rely on well-known string distance measures (e.g., Levenshtein) while type similarity relies on the approach in [24] mapping to the compatibility level between types. The local overall distance would then equal the weighted sum of the two different distances.

If all input parameter parts are matched, the compared messages are semantically compatible. Otherwise, the compared services are semantically incompatible.

Let us continue the running example to explain syntactic matchmaking. Suppose that A & C were found equivalent. C maps to message type MT_1 containing 4 information pieces MT_{11} , MT_{12} , MT_{13} and MT_{14} . A maps to message type MT_2 containing 3 information pieces MT_{21} , MT_{22} , and MT_{23} .

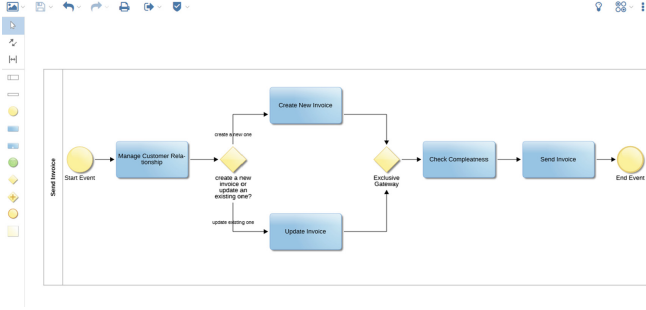


Fig. 3. The Send Invoice business process in BPMN 2.0

Based on matching message types to ontology concepts, we have that MT_{11} and MT_{21} map to $A.A1$ while MT_{12} and MT_{22} map to $A.A2$. Thus, the information pieces are transformed into $\{A.A1, A.A2, MT_{13}, MT_{14}\}$ for first message type and $\{A.A1, A.A2, MT_{23}\}$ for the second. For those pieces not mapping to ontology attributes, we solve a bipartite matching problem again. Suppose that $dist(MT_{13}, MT_{23}) = 0.8$ and $dist(MT_{14}, MT_{23}) = 0.2$. Then, the sole mapping to be selected will be $\{MT_{13} \rightarrow MT_{23}\}$. If we replace MT_{23} with MT_{13} , we then need to check whether $\{A.A1, A.A2, MT_{13}\}$ is subset of $\{A.A1, A.A2, MT_{13}, MT_{14}\}$ which holds.

6 Validation

Our approach was validated based on a use case developed by CloudSocket’s industrial partners. We focused on a very common BPs among SMEs - the Send Invoice one. This BP is modelled in BPMN, see Fig. 3, via our BPaaS Design environment. It starts with the “Manage Customer Relationship” activity; next an exclusive gateway splits the BP flow between either creating a new invoice or updating an existing one. Then, invoice completeness is checked, and finally the invoice is sent. Subsequently, starting with this BPMN process, we acquaint the reader with a prerequisite plus the main steps involved in our approach.

Prerequisite Step: Service Profile Registration. The following services were inserted in the *KB* as instances of *CloudService* class:

- YMENS, Zoho and Sugar CRM were inserted as CRM systems which were annotated with the action *Manage*, the object *Customer* and the APQC category *3.5.2.4 Manage Customer Relationship*
- Mathema Document Generator, Open Source Billing, Simple Invoice and InvoiceNinja as invoicing systems annotated with action *Generate*, object *Invoice* and APQC category *9.2.2.2 Generate Customer Billing Data*
- Gmail, Ninja_email and Mailjet were inserted as e-mail systems which were annotated with the action *Manage*, the object *Invoice* and the APQC category *9.2.2.3 Transmitting Billing Data to Customers*

Table 1 shows a part of the non-functional profiles of the considered services.

Table 1. Functional requirements for each group and single activity

| Service | Monthly downtime | Response time level | File type | No of simul. users | Execution constraint |
|---------------------|------------------|---------------------|-------------------------------|--------------------|----------------------|
| YMENS CRM | 4 min | High | Office doc, PDF, audio, video | 500 | None |
| Zoho CRM | 4 min | High | Office doc, PDF, audio, video | 500 | None |
| Sugar CRM | 10 min | Medium | Office doc, PDF, audio, video | 200 | 500 (monthly basis) |
| InvoiceNinja | 4 min | High | Office doc, PDF, audio, video | 600 | None |
| Ninja Email | 4 min | High | Office doc, PDF, audio, video | 400 | None |
| Simple Invoices | 10 min | Medium | Office doc, PDF, audio, video | 300 | None |
| Mailjet | 4 min | High | Office doc, PDF, audio, video | 100 | 1K (monthly basis) |
| Open Source Billing | 4 min | Medium | Office doc, PDF, audio, video | 200 | None |
| Gmail | 4 min | High | Office doc, PDF, audio, video | 100 | None |

First Main Step: Business Matchmaking. *BM* was used to identify the most suitable cloud services. As a first step, the questionnaire was applied on the whole BP (see starting notebook at Fig. 4).

We specified functional requirements in the first 3 questions - object *Send*, action *Invoice* and APQC category 9.2.2 *Invoice Customer* - and none of the cloud services matched.

Next, the questionnaire was applied on two single activities (i.e., Manage Customer Relationship and Send Invoice) as well as on a group of activities (i.e., Create Invoice, Update Invoice and Check Invoice Completeness).

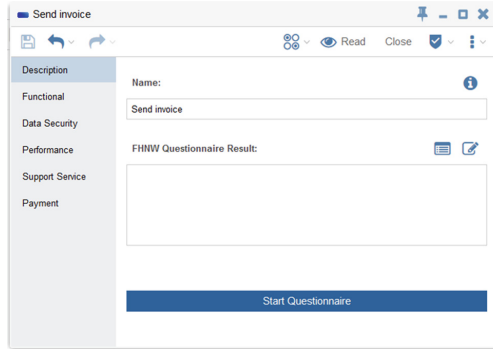


Fig. 4. The starting notebook for the whole process

Table 2 shows the functional requirements for each activity/group. In the first case, after specifying action, object and APQC category, the questionnaire showed the 3 matching cloud services: YMENS, Zoho and SugarCRM. In the 4th question, we chose the *Performance* category, and the question prioritisation algorithm kicked in. The question regarding the number of simultaneous users was asked (attribute with highest entropy) and a value of 500 was entered. This filtered out SugarCRM as it has the capability of max 200 simultaneous users.

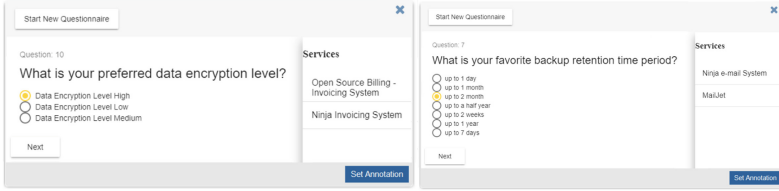
Table 2. Functional requirements for each group and single activity

| BPMN activity | Action | Object | APQC category |
|---|----------|-----------------------|--|
| Manage Customer Relationship (<i>Single activity</i>) | Manage | Customer Relationship | 3.5.2.4 Manage Customer Relationship |
| Send Invoice (<i>Single activity</i>) | Send | Invoice | 9.2.2.3 Transmit Billing Data to Customers |
| Create New Invoice, Update Invoice, Check Invoice Completeness (<i>Group of activities</i>) | Generate | Invoice | 9.2.2.2 Generate Customer Billing Data |

Similarly, we applied the questionnaire on the designated group of activities. The matching services were InvoiceNinja and Open Source Billing, see Fig. 5a.

Finally, we applied the questionnaire on the last BP activity: Send Invoice. The matching cloud services were Ninja E-mail and Mailjet (see Fig. 5b).

Second Main Step: Technical Matchmaking & Selection. As the final result maps to two services per each activity (group), we now proceed with the technical matching and selection. Suppose that the user provides the next global



(a) The selected invoice management services (b) The selected email services

Fig. 5. The selected services for last two activity groups

requirements for the whole process: $cost < 100$ euros per month, $cycletime < 1$ min and $VPM < 16$ (#vulnerabilities per month). Further, suppose that the user imposes for the *Manage Customer Relationship* activity the following constraints: $responsetime < 30$ s and $VPM < 10$. Finally, Table 3 depicts the non-functional profiles of the remaining services.

Table 3. The technical non-functional offerings of the 6 services

| Service | Cost | Response time | VPM |
|---------------------|----------|---------------|-----|
| ZOHO CRM | 30 euros | 35 s | 10 |
| YMENS CRM | 35 euros | 20 s | 05 |
| Mailjet | 25 euros | 10 s | 02 |
| Ninja_Email | 10 euros | 15 s | 03 |
| Open Source Billing | 35 euros | 25 s | 08 |
| InvoiceNinja | 45 euros | 10 s | 05 |

Technical non-functional matching would then filter Zoho CRM as it does not conform to the local constraints posed for the CRM activity. This leads to selecting over 4 solutions as we have one candidate for the first (group) of activities and 2 candidates for the rest two activity groups. However, while running service selection, it is detected that the Ninja.Email and Open Source Billing are incompatible, which leaves us with 3 solutions. Moreover, the solution mapping to selecting YMENS, Open Source Billing and Ninja.Email has VPM equal to 16 violating the respective global constraint. So, in the end, we need to select between 2 solutions which are depicted in Table 4.

Table 4. The final ordered solutions produced

| Solution | Cost | Cycle time | VPM | Utility |
|---------------------------------------|----------|------------|-----|---------|
| YMENS + InvoiceNinja + Ninja.Email | 90 euros | 45 s | 13 | 0.144 |
| YMENS + Open Source Billing + Mailjet | 95 euros | 50 s | 15 | 0.099 |

As the broker requires to optimise all non-functional terms (cost, cycle time and VPM), it gives equal preference over them. By also considering that the activities are sequentially executed in the BPaaS workflow, the final result would map to selecting services YMENS, InvoiceNinja, and Ninja_Email. While there is perfect syntactic compatibility between InvoiceNinja and Ninja_Email as they are offered by the same company, in the case of YMENS CRM and InvoiceNinja the message types are compatible but still need to be aligned (e.g., attributes *accountid* and *id_number* mapping to the same attribute *id* of concept *Client*). As such, the MS service was included between these 2 services resulting in a workflow with 4 services sequentially executed (YMENS CRM \rightarrow MS \rightarrow InvoiceNinja \rightarrow Ninja_Email).

7 Conclusions and Future Work

This paper has introduced a novel architecture for the design of BPaaS products able to effectively deal with the business-to-IT alignment problem in order to map an initial domain-specific BP into an executable BPaaS workflow. Such an architecture has been carefully designed and implemented to include suitable components which focus on different parts of the business-to-IT alignment problem, including business and technical matchmakers, a service selection as well as an automatic workflow update component to enable the effective addressing of the message compatibility problem in service-based workflow execution.

Our future work will focus on more advanced research challenges which include: (a) the automatic production of a more complete and more close to production workflow via the incorporation of different kinds of non-service tasks (see previous section); (b) the automatic population of the KB; (c) the coverage of additional cases in business-to-technical-requirement alignment.

Acknowledgments. This research has received funding from the European Community's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

References

1. Antunes, G., Bakhshandeh, M., Borbinha, J., Cardoso, J., Dadashnia, S., Francescomarino, C.D., Dragoni, M., Fettke, P., Gal, A., Ghidini, C., Hake, P., Khiat, A., Klinkmüller, C., Kuss, E., Leopold, H., Loos, P., Meilicke, C., Niesen, T., Pesquita, C., Péus, T., Schoknecht, A., Sheerit, E., Sonntag, A., Stuckenschmidt, H., Thaler, T., Weber, I., Weidlich, M.: The process model matching contest 2015. In: Lecture Notes in Informatics (2015)
2. Azzini, A., Braghin, C., Damiani, E., Zavatarelli, F.: Using Semantic Lifting for Improving Process Mining: A Data Loss Prevention System Case Study (2013)
3. Benaboud, R., Maamri, R., Sahnoun, Z.: Agents and owl-s based semantic web service discovery with user preference support. *Int. J. Web Semant. Technol.* **4**(2), 57–75 (2013)

4. Cliffe, O., Andreou, D.: Service Matchmaking Framework. Public Deliverable D5.2a, Alive EU Project Consortium, 10 September 2009. http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=28&Itemid=49
5. Cloud Select Industry Group (C-SIG): Cloud Service Level Agreement Standardization Guidelines. Technical report, EC (2014)
6. Duipmans, E.: Business Process Management in the cloud: Business Process as a Service (BPaaS). Technical report (2012)
7. Emmenegger, S., Hinkelmann, K., Laurenzi, E., Martin, A., Thönssen, B., Witschel, H.F., Zhang, C.: An ontology-based and case-based reasoning supported workplace learning approach. In: Hammoudi, S., Pires, L.F., Selic, B., Desfray, P. (eds.) MODELSWARD 2016. CCIS, vol. 692, pp. 333–354. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66302-9_17
8. Emmenegger, S., Hinkelmann, K., Laurenzi, E., Thönssen, B.: Towards a procedure for assessing supply chain risks using semantic technologies. In: Fred, A., Dietz, J.L.G., Liu, K., Filipe, J. (eds.) IC3K 2012. CCIS, vol. 415, pp. 393–409. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-54105-6_26
9. Faria, D., Pesquita, C., Santos, E., Cruz, I.F., Couto, F.M.: AgreementMakerLight Results for OAEI 2013. http://disi.unitn.it/~p2p/OM-2013/oaei13_paper1.pdf
10. Frank, U.: Domain-specific modeling languages: requirements analysis and design guidelines. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., Bettin, J. (eds.) Domain Engineering, pp. 133–157. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36654-3_6
11. Hinkelmann, K., Gerber, A., Karagiannis, D., Thoenssen, B., van der Merwe, A., Woitsch, R.: A new paradigm for the continuous alignment of business and IT: Combining enterprise architecture modelling and enterprise ontology. *Comput. Ind.* **79**, 77–86 (2016)
12. Hinkelmann, K., Kritikos, K., Kurjakovic, S., Lammel, B., Woitsch, R.: A modelling environment for business process as a service. In: Krogstie, J., Mouratidis, H., Su, J. (eds.) CAiSE 2016. LNBIP, vol. 249, pp. 181–192. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39564-7_18
13. Hinkelmann, K., Laurenzi, E., Lammel, B., Kurjakovic, S., Woitsch, R.: A semantically-enhanced modelling environment for business process as a service. In: ES, pp. 143–152. IEEE (2016)
14. Hrgovic, V., Karagiannis, D., Woitsch, R.: Conceptual modeling of the organisational aspects for distributed applications: the semantic lifting approach. In: COMPSAC Workshops, pp. 145–150. IEEE (2013)
15. Hwang, C., Yoon, K.: Multiple criteria decision making. In: *Lecture Notes in Economics and Mathematical Systems* (1981). <https://doi.org/10.1007/978-3-642-48318-9>
16. Jiang, S., Aagesen, F.A.: An approach to integrated semantic service discovery. In: Gaiiti, D., Pujolle, G., Al-Shaer, E., Calvert, K., Dobson, S., Leduc, G., Martikainen, O. (eds.) AN 2006. LNCS, vol. 4195, pp. 159–171. Springer, Heidelberg (2006). https://doi.org/10.1007/11880905_14
17. Karagiannis, D., Buchmann, R.A., Burzynski, P., Reimer, U., Walch, M.: Fundamental conceptual modeling languages in OMiLAB. In: Karagiannis, D., Mayr, H., Mylopoulos, J. (eds.) Domain-Specific Conceptual Modeling, pp. 3–30. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_1
18. Klusch, M.: Semantic web service coordination. In: Schumacher M., Schuldt H., Helin H. (eds.) CASCOS: Intelligent Service Coordination in the Semantic Web, pp. 59–104 (2008). https://doi.org/10.1007/978-3-7643-8575-0_4

19. Kritikos, K., Magoutis, K., Plexousakis, D.: Towards knowledge-based assisted IaaS selection. In: CloudCom, pp. 431–439. IEEE Computer Society (2016)
20. Kritikos, K., Plexousakis, D.: Requirements for QoS-based web service description and discovery. *IEEE Trans. Serv. Comput.* **2**(4), 320–337 (2009)
21. Kritikos, K., Plexousakis, D.: Novel optimal and scalable nonfunctional service matchmaking techniques. *IEEE Trans. Serv. Comput.* **7**(4), 614–627 (2014)
22. Kritikos, K., Plexousakis, D.: Multi-cloud application design through cloud service composition. In: CLOUD, pp. 686–693. IEEE, New York (2015)
23. Kritikos, K., Plexousakis, D.: Towards combined functional and non-functional semantic service discovery. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol. 9846, pp. 102–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_7
24. Kritikos, K., Plexousakis, D., Paternò, F.: Task model-driven realization of interactive application functionality through services. *TiiS* **3**(4), 25 (2014)
25. Laurenzi, E., Hinkelmann, K., Reimer, U., van der Merwe, A., Sibold, P., Endl, R.: DSML4PTM - A domain-specific modelling language for patient transferal management. In: ICEIS 2017, Porto, Portugal, pp. 520–531 (2017)
26. Plebani, P., Pernici, B.: URBE: web service retrieval based on similarity evaluation. *IEEE Trans. Knowl. Data Eng.* **21**(11), 1629–1642 (2009)
27. Saati, T.: *The Analytic Hierarchy Process*. McGraw-Hill, New York (1980)
28. Silver, B.: *BPMN Method and Style*, 2nd edn. Cody-Cassidy Press, Aptos (2011)
29. Uschold, M., King, M., Morale, S., Zorgios, Y.: The enterprise ontology. *Knowl. Eng. Rev.* **13**(01), 31–89 (1998)
30. Vaishnavi, V., Kuechler, B.: Design Science Research in Information Systems (2004). <http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf>
31. Watfa, M.K., Najjar, N.A.L., Cheikha, J., Buali, N.: A new framework for cloud business process management. In: Zhang, Y., Peng, L., Youn, C.-H. (eds.) Cloud-Comp 2015. LNICST, vol. 167, pp. 83–92. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38904-2_9
32. Whibley, P.: *BPM in the Cloud - Transforming the Business Case for Process Improvement*. Technical report (2012)
33. Woitsch, R., Utz, W.: Business Processes as a Service (BPaaS): A Model-Based Approach to align Business with Cloud offerings (2015). <https://zenodo.org/record/35583#.WbDscNhLfmE>

CloudWays

Preface of CloudWays 2017

Cloud computing has been the focus of attention of both academic research and industrial initiatives. From a business point of view, organizations can benefit from the on-demand and pay-per-use model offered by cloud services rather than an upfront purchase of costly and over-provisioned infrastructure. From a technological perspective, the scalability, interoperability, and efficient (de-)allocation of resources through cloud services can enable a smooth execution of organizational operations.

Regardless of the benefits of cloud computing, many organizations still rely on business-critical applications in the form of legacy systems that have been developed over a long period of time using traditional development methods. Despite often serious maintainability issues, (on-premise) legacy systems are still crucial as they support core business processes. Therefore, migrating legacy systems towards cloud-based platforms allows organizations to leverage their existing systems deployed and provided (using publicly available resources) as scalable cloud services.

This third edition of the workshop – the Third International Workshop on Cloud Adoption and Migration (CloudWays 2017) – was held in Oslo, Norway, on September 27, 2017, as an ESOC satellite event. The first edition was held in September 2015 in Taormina, Italy, and the second in September 2016 in Vienna, Austria, both also as a satellite events of ESOC. The workshop’s goals were: to bring together cloud migration and cloud architecture experts from both academia and industry; to promote discussions and collaboration among participants; to help disseminate novel cloud adoption, migration, and architecture practices and solutions; and to identify future cloud challenges and dimensions that help software applications to be architecture for and deployed in the cloud.

In this third edition, six full papers were accepted for presentation during the workshop, out of a total of nine submissions.

The first paper “Engineering Cloud-Based Applications: Toward an Application Life Cycle” by Vasilios Andrikopoulos aims to distill the challenges of adopting and architecting cloud-based applications into a life cycle framework that takes cloud characteristics such as service orientation, distribution, multi-tenancy, and utility computing into account.

The second paper “A Cloud Computing Workflow for Managing Oceanographic Data” by Salma Allam, Antonino Galletta, Lorenzo Carnevale, Moulay Ali Bekri, Rachid El Ouahbi, and Massimo Villari was the first in the workshop to focus on data aspects, which was done through the discussion of workflow concerns in the context of an oceanography use case.

The third paper “Pattern-Driven Architecting of an Adaptable Ontology-Driven Cloud Storage Broker” by Divyaa Manimaran Elango, Frank Fowley, and Claus Pahl looks at interoperability in cloud computing. A cloud service broker is introduced from

a software design perspective, looking at architecture and design patterns used in the construction.

The fourth paper “Cloud-Native Databases: An Application Perspective” by Josef Spillner, Giovanni Toffetti Carughi, and Manuel Ramírez López continues the data focus. Here databases as a services are investigated as an architectural concern by looking at cloud-nativeness as a property.

The fifth paper “Using a Cloud Broker API to Evaluate Cloud Service Provider Performance” by Divyaa Manimaran Elango, Frank Fowley, and Claus Pahl reports on performance testing and comparison of different storage services. A broker API is used to monitor and compare the different services.

The final paper “TosKer: Orchestrating Applications with TOSCA and Docker” by Antonio Brogi, Luca Rinaldi, and Jacopo Soldani is also concerned with interoperability. Raising the abstraction level through standards, languages such as the orchestration language TOSCA is used to manage Docker containers.

In addition to the presentation of the accepted papers, an invited talk titled “Business Processes and Smart Devices — A Marriage of Convenience?” was jointly organized with participants of the BPM@Cloud workshop focusing on the challenges and perspectives with process modelling in the cloud, looking specifically also at edge and IoT as a context. The presentation was given by Prof. Pierluigi Plebani from the Politecnico di Milano, Italy.

We take this opportunity to thank all authors, members of the Program Committee, and workshop attendees, whose participation was invaluable to the success of the event. We also acknowledge the support provided by The Irish Centre for Cloud Computing and Commerce (IC4) and the Free University of Bozen-Bolzano (UniBZ).

Claus Pahl
Nabor Mendonça
Pooyan Jamshidi


Organization

Program Committee

| | |
|-----------------------------|--|
| Aakash Ahmad | IT University of Copenhagen, Denmark |
| Vasilios Andrikopoulos | University of Stuttgart, Germany |
| William Campbell | Birmingham City University, UK |
| Vinicius Cardoso Garcia | Federal University of Pernambuco, Brazil |
| Fei Cao | University of Central Missouri, USA |
| Nabil El Ioini | Free University of Bozen-Bolzano, Italy |
| Nicolas Ferry | SINTEF, Norway |
| Frank Fowley | Dublin City University, Ireland |
| Sören Frey | Daimler TSS, Germany |
| Wilhelm (Willi) Hasselbring | Kiel University, Germany |
| Pooyan Jamshidi (Co-chair) | Carnegie Mellon University, USA |
| Ali Khajeh-Hosseini | AbarCloud, UK |
| Xiaodong Liu | Napier University, Edinburgh, UK |
| Theo Lynn | Dublin City University, Ireland |
| Paulo Henrique Maia | State University of Ceará, Brazil |
| Nabor Mendonça (Co-chair) | University of Fortaleza, Brazil |
| Claus Pahl (Co-chair) | Free University of Bozen-Bolzano, Italy |
| Dana Petcu | West University of Timisoara, Romania |
| Alessandro Rossini | EVRY Cloud Services, Norway |



Engineering Cloud-Based Applications: Towards an Application Lifecycle

Vasilios Andrikopoulos^(✉) 

Johann Bernoulli Institute for Mathematics and Computer Science,
University of Groningen, Groningen, The Netherlands
v.andrikopoulos@rug.nl

Abstract. The adoption of cloud computing by organizations of all sizes and types in the recent years has created multiple opportunities and challenges for the development of software to be used in this environment. In this work-in-progress paper, the focus is on the latter part, providing a view on the main research challenges that are created for software engineering by cloud computing. These challenges stem from the inherent characteristics of the cloud computing paradigm, and require a multi-dimensional approach to address them. Towards this goal, a lifecycle for cloud-based applications is presented, as the foundation for further work in the area.

Keywords: Cloud computing · Software engineering
Cloud-based applications · Software lifecycle

1 Introduction

The adoption of cloud computing has increased dramatically since the introduction of the term only roughly ten years ago—despite the fact that the technologies underpinning the paradigm have been around for a while longer. It is not an exaggeration to claim that in one way or another cloud computing offerings and associated technologies are currently being used by the majority of software-intensive enterprises. A report of the Thoughtworks Technology Advisory Board back in May 2015¹, for example, claims that “*Organizations have accepted that “cloud” is the de-facto platform of the future, and the benefits and flexibility it brings have ushered in a renaissance in software architecture.*” From the thousand professionals from across sectors participating to RightScale’s annual survey in early 2017 [31], 95% are reporting that the organization they belong to is already using or experimenting with the use of cloud computing.

Under the umbrella of the same term, however, there are multiple service delivery and deployment models on offer, succinctly summarized by NIST’s widely accepted definition of cloud computing [24]. The availability of these

¹ Thoughtworks Tech Radar, May 2015: <https://assets.thoughtworks.com/assets/technology-radar-may-2015-en.pdf>.

options, in conjunction with the plethora of offerings by cloud providers like Amazon Web Services (AWS), Microsoft Azure (MSA), and Google Compute Platform (GCP), and software solutions for the deployment of private clouds such as the ones from VMware and OpenStack, create both opportunities and challenges for software developers [4]. Even the process of selecting an appropriate provider to run software on is an open research subject, with many of the issues identified in [34] (e.g. lack of standardization in the QoS descriptions and lack of long term performance prediction) still valid today. As such, there are still many issues that need to be resolved with respect to how cloud computing is to be used for software development.

At the same time, in the recent years the discourse on the best practices and principles of software development, at least in the industry, has been affected significantly by the introduction of two movements that have a co-dependence relation with cloud computing. The first one is the use of DevOps technologies and processes in order to bridge the gap between development and operations of software [8] in order to streamline software delivery and maintenance. The adoption of Continuous Delivery/Integration (CD/CI) techniques with frameworks like Jenkins² used together with deployment automation tools like Chef³ or Ansible⁴ shortens the development cycle dramatically and produces synergy with agile-oriented software development practices. Allowing for the management of multiple software stacks running in partially isolated containers inside one operating system as made popular by Docker⁵, is the logical extension of this approach: each architectural component is developed, deployed, managed, and updated in its own software stack, and therefore it can follow a life cycle that is loosely coupled with the overall system evolution. This principle is made even more prominent in the second of the movements relevant to the discussion, i.e. microservices [27]. While there is an ongoing discussion in the academic community related to the actual innovation of microservices in comparison to Software-Oriented Architecture, it is important to notice how the notion of microservices have integrated into practice the use of design patterns, that so far have been mostly adopted at a much lower level (e.g. the Gang of Four book). Entries on microservices in Martin Fowler's blog⁶, a popular grey literature source for practitioners and researchers provides many instances of this phenomenon.

In summary, therefore, the virtualization of resources and their offering as services, in conjunction with the DevOps movement, the containerization of software stacks, and the use of microservices, have evolved the way that software is developed, deployed, and managed over time. The key message of this paper is that *engineering software, and in particular software architecture, should similarly evolve*. For the purposes of scoping, the discussion is focused

² Jenkins: <https://jenkins.io/>.

³ Chef: <https://www.chef.io/>.

⁴ Ansible: <https://www.ansible.com/>.

⁵ Docker: <https://www.docker.com/>.

⁶ For example: Microservices, by James Lewis and Martin Fowler (March 2014): <https://martinfowler.com/articles/microservices.html>.

on how software engineering can change to incorporate cloud-related concepts by means of introducing a *cloud-based application lifecycle*. In the absence of a widely accepted definition of what constitutes one, and following the definition of service-based applications discussed in [3], this paper uses a working definition of *Cloud-based applications (CBAs)* as *applications that rely on one or more cloud services in order to be able to deliver their functionality to their users*. CBAs therefore include both cloud-enabled through migration [2] and cloud-native applications [21].

The rest of this paper is structured as follows: Sect. 2 identifies and presents the most relevant challenges to cloud-based application engineering (definitely not an exhaustive list). Section 3 transforms these challenges into a set of requirements on lifecycle methodologies in this context. Consequently, Sect. 4 discusses a CBA lifecycle that aims to address these requirements as the basis for future research. Finally, Sect. 5 compares the proposed lifecycle with related approaches, and Sect. 6 concludes with a short summary and future work.

2 Major Challenges

Following the NIST definition [24], cloud computing has the following essential characteristics: (i) *On-demand self-service*: appropriate interfaces are offered to consumers to access resources (computational, storage, network, etc.) in an automated manner. (ii) *Broad network access*: resources are accessed over the network by heterogeneous clients. (iii) *Resource pooling*: service providers are enforcing a multi-tenant model of sharing the offered resources. (iv) *Rapid elasticity*: the volume of accessed resources can be adjusted dynamically, by any quantity and at any time. (v) *Measured service*: a metering mechanism is used to ensure appropriate billing for the used resources in predefined periods of time.

The combinations of these characteristics has severe implications for the software that is being developed in this environment. In the following we identify four major challenges that arise due to these characteristics.

2.1 *aaS Software Model

The first major challenge stems from the fact that resources are offered in the *Everything as a Service (*aaS)* model, usually affiliated with the categorization of delivery models into Infrastructure (IaaS), Platform (PaaS), and Software as a Service (SaaS), also covered by the NIST definition. The *aaS model is a natural outcome of the first two characteristics (i.e. *on-demand self-service*, and *broad network access*) and in many cases manifests as sets of RESTful APIs that are exposing cloud resources through relatively simple CRUD operations. While there has been lots of work on the subject of engineering service-based applications in the last 15 or so years, see for example [3], the very nature of service orientation still poses particular difficulties when used as the model for accessing resources. These can be attributed to the following:

Information Hiding Behind Interfaces: Exposing only the amount of information that is absolutely necessary for clients to use a service is one of the fundamental premises of service orientation [12]. However, this means that software developers have to refer to documentation and help desks in order to understand the boundary conditions and assumptions of consuming each resource.

Lack of Control and Observability over Resource Implementation: While the on-demand self-service characteristic prescribes a degree of control over the consumed resources by removing the need for administration on the part of the provider, this control is in practice limited to the operations defined in the service API, that for all practical purposes act as black box endpoints.

Distributed and Heterogeneous Environment: Distribution transparency [33] is an essential feature of offered services, creating an impression of homogeneity and opaqueness to software developers. Nevertheless, the operating environment is fundamentally distributed, irrespective of the type of software developed on it (distributed or not).

Evolution Driven by 3rd Parties: As with many other API publishers in the past, cloud providers reserve the right to change their supported APIs at any point in time—and they do so for various reasons. As such, therefore, the evolution of software developed on these solutions is at least partially driven by the cloud providers and beyond software developer control.

Lastly, it can also be argued that while it is indeed possible to build all kinds of systems on top of cloud resources, it is consistent with the model that it is offered to design and implement them as services themselves. Doing so, however, imposes its own challenges, as evidenced by the continuous research output of the SOA community in the last two decades. The most thorny issue to deal with is probably the design of the system as services itself; indicative of the complexity of this issue is the fact that service design is identified as a major research question in both the SOA research roadmap [30], and its revision ten years later [10]. Further work towards this direction is therefore required.

2.2 Multi-tenancy of Resources

One of the most difficult challenges to address, especially for performance-sensitive systems, is that of the shared nature of cloud resources due to its *resource pooling* characteristic. In a sense it is exactly this characteristic which makes rapid elasticity possible, while allowing for resource prices to be offered at very low levels, as also discussed by the next challenge. In essence, multiple tenants sharing the same infrastructure enable economies of scale for service providers and allow for higher utilization on the provider side through smart scheduling of large volumes of work load.

This sharing of resources, however, leads at the same time to performance variability that is external to the application itself, and as such outside of the

control of the system developer. The inherent variance of cloud offerings has been documented in a series of publications: in [22], for example, large deviations are reported for similar in specification offerings across different providers, while significant variance can be observed in the same provider and offering within the same day and week [32] (and even more so across different availability zones), or even over the period of a year for the same offering [17]. Benchmarking cloud applications is faced with multiple challenges, see for example [9, 14], and is not readily available as a tool for software developers to incorporate in their toolset. Cloud monitoring [1] is therefore the most common way to check and potentially address detrimental performance variation of the consumed resources.

2.3 Utility Computing

One of the main reasons for the wide adoption of cloud computing is the transfer of costs from the capital to operating expenses through its “pay as you go” model [6], enabled by its *measured service* characteristic. In this sense, cloud computing can be seen as an implementation of the utility computing vision [39]. Access to computational resources in this context is enabled in a utility-oriented model, and results in the illusion of virtually infinite resources being available—assuming of course a sufficiently large budget [6]. At the same time, the use of economies of scale on behalf of the service providers, and the environment of intense competition for a very lucrative market, result into continuously decreasing prices for the offered resources. This creates the dynamics of a “race to zero” phenomenon, especially in storage offerings⁷. Even if the provider prices are not lower in comparison with operating one’s own data center as e.g. in the (already outdated) analysis of [37], there are boundary conditions that still make the use of cloud solutions favorable to the alternative [38]. The key is in the *rapid elasticity* characteristic which allows for quick scaling to cope with dynamic demand, resulting in compensation of potentially incurred losses throughout relatively stable demand periods by means of serving requests that would otherwise be over capacity and therefore resulting in loss of revenue.

Nevertheless, cheap is not the same as free of charge, and costs for successful cloud-based companies might run so high that result in their profit margin shrinking to the point of necessitating the migration to their own data centers instead, as documented by the case of Dropbox⁸, a company that was famous for running all their infrastructure on Amazon Web Services until that point. Rightscale’s 2017 State of the Cloud survey [31] reports two stark findings that are relevant to this discussion: first, mature adopters of the technology are more concerned with cost management in comparison to beginners to it; second, only a minority of companies actually take measures to minimize unnecessary costs (e.g. VMs unnecessarily being active). Some notion of costs control is therefore clearly necessary.

⁷ See for example: <http://www.computerweekly.com/microscope/news/4500271376/Whatever-the-cost-may-be-Cloud-price-war-continues>.

⁸ See <https://www.wired.com/2016/03/epic-story-dropboxs-exodus-amazon-cloud-empire/>.

2.4 Distributed Topology

There is no escaping the fact that systems developed in the cloud environment are essentially distributed, and they need to be designed, implemented, and operated as such [11]. Distribution in this case is both spatial and logical, but distribution transparency [33] is partially violated when e.g. availability zones are used for the deployment of applications. On top of this, there are multiple offerings by service providers that can be used as alternatives to application components [4] taking advantage of the *on-demand self service* characteristic. For example, Database as a Services (DBaaS) offerings can replace completely the data layer of an application, providing natively scaling mechanisms to cope with increasing demand. An illustration of the range of possibilities available to software architects is the case of Netflix, which combines AWS EC2, S3, EBS and other offerings to run in a cloud-only environment⁹.

Adding to the size of the design space is the capability to use containers as the means for enabling portability of application components and work loads across cloud providers, essentially expanding on the characteristic of *resource pooling*. In conjunction with a cloud orchestration layer, containers allow for a series of benefits like reduced (infrastructure) complexity, automation of portability, better governance and security management, transparent geographical domain-aware distribution, and the ability to automate services that offer policy-based optimization and self-configuration [23]. As a result, there are many possible system configuration options that are optimal under different dimensions [4], e.g. cost versus performance, creating exceptional challenges to software architects in identifying the best solution for their needs.

3 Requirements on the Solution Space

From the discussion above it becomes quickly obvious that addressing these challenges is a multi-faceted undertaking, and that their nature requires them to be considered throughout the lifecycle of software systems operating in the Cloud. The following constraints are, as a result, imposed on possible solutions for engineering *cloud-based applications (CBAs)*:

1. Irrespective of the purpose and type of software under consideration, cloud-based application development should *understand and incorporate service-orientation concepts*. In practice, this means that resources are accessed through programmatic interfaces, which in turn favors the Infrastructure as a Code approach [16] that homogenizes the way that the software itself and its supporting infrastructure is managed. Across similar lines, cloud service composition [18], which deals with the selection and aggregation of cloud services in order to support software, needs to be considered on equal grounds with (software) service composition [30] which delivers functionality by combining independent services.

⁹ Netflix Global Cloud Architecture: <https://www.slideshare.net/adrianco/netflix-global-cloud>, slide 26.

2. System design should incorporate the notion of *dynamic topology*. Topology here refers to the software and infrastructure stack required to operate the software artifacts under consideration, including e.g. the middleware associated with them. The system topology is prone to change over time due to changes (a) in the system architecture, and (b) refactoring of the infrastructure that supports the system. This might also include incorporation of new services by the same cloud provider or migration to another provider and/or deployment model. In this sense, the resolution of system architecture into concrete deployment models should rely on the generation of viable topologies through e.g. graph transformations, as per [4], instead of explicit modeling of alternatives. This is a consequence of the very large amount of available alternatives during design when considering all the different configurations available for each service type.
3. *Self-* characteristics* (e.g. self-management, -adaptation, -healing, -configuration, etc.) are necessary to deal with the multi-tenancy induced performance variability and its impact to the QoS of cloud-based applications. The introduction of a MAPE-K (Monitor, Analyse, Plan, and Execute over a Knowledge base) feedback loop [20] is a necessary and very common solution at this level as the means to implement control [29], but the difficulty is in evaluating the impact of individual cloud services, e.g. a DBaaS solution, to overall performance. Furthermore, the connection between run-time observations and design-time predictions is not sufficiently covered by the state of the art [15], and further work is necessary towards this direction. End-to-end performance measurement is potentially more important—alternative viable topologies have to be evaluated after all against their actual effectiveness in generating revenue—and in case of software delivered as services relatively easy to implement.
4. An *awareness of consumed resources* on self-management level during both development and operation of the system is essential. Cost models that cover the various deployment models, e.g. an extension of the model for hybrid clouds discussed in [19], should be used for this purpose. However, such analysis cannot be only performed offline. Instead, design- and run-time cost analysis should complement each other [25], resulting in cost models that are dynamically updated by actual billing data received from the cloud provider.

In the following we introduce a lifecycle model for cloud-based applications that incorporates the constraints discussed above as the means for defining in the future a holistic framework for engineering cloud-based applications. For this purpose the lifecycle model of service-based applications as discussed in [3] is used as the inspiration for this work.

4 Cloud-Based Applications Lifecycle

4.1 The Phases of the Lifecycle

Figure 1 illustrates the proposed lifecycle of CBAs. Before proceeding with explaining the stages of the lifecycle, it needs to be pointed out that for the

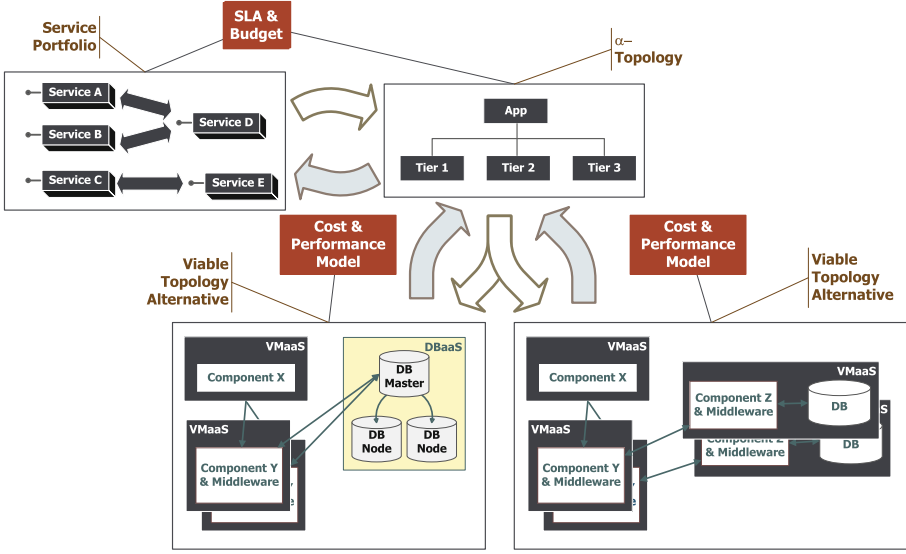


Fig. 1. The lifecycle of cloud-based applications

purposes of this discussion, there is no clear design- and run-time (or development and operation, respectively) distinction, but more of a spectrum of activities spanning between them. The everything as a service and dynamic topology challenges affect more the one end (design), while performance variability and cost awareness more the other (run time). However it is impractical to attempt to assign them to specific stages of the lifecycle. The proposed CBA lifecycle (as shown in the figure) reflects this by intentionally not identifying when the transitions between stages are to take place, but only the transition relations between them. In this respect, the presented lifecycle is in accordance with the main principles of the DevOps movement [8] which unifies the different stages of software lifecycle.

Looking now at the figure, and starting from its top left part, the highest stage of the lifecycle consists of the *service portfolio* for the application, i.e. the collection of services that implement the functionalities offered by the application. Such services could be composed out of other services, belonging either to the same portfolio, or being external to it, as per the well established SOA practice [30]. Following the same principles, the service portfolio is the outcome of a *service identification* phase that connects higher level requirements and business operations into functionalities to be exposed by the application as services. In terms of how these services are mapped into software components and its supporting middleware, a *decomposition* into structural tiers can be applied using one of the methodologies discussed in [3]. In principle, non-application specific software components should be excluded from this process, resulting into a system architecture expressed as a set of α -topologies [4] (top center of Fig. 1). The intentional exclusion of the underlying software stack from this stage (except

where it cannot be avoided as e.g. in the case of customized middleware that needs to be rolled out together with the application) allows for flexibility in the transition to the next stage, that of *viable topology alternatives*, each one of which represent the whole software stack and its relation to the application components (bottom half of Fig. 1). Viable topology models encapsulate the various types of cloud services (e.g. VM or DB as a Service in the figure) that are part of the infrastructure supporting the software stack of the application.

As discussed above, and due to the numerous cloud service offerings currently available, a large number of viable topology alternatives potentially exist for each application. Selecting between them can be, and usually is interpreted as an optimization problem for which there are many techniques available (see [4] for further discussion). However, an alternative approach would be to look into this situation as an exploratory search problem instead. In this context, identifying a unique optimal solution in advance would be of not such interest as in transitioning between different alternative solutions in order to identify the optimal for the current conditions. For this purpose, the overall consumer utility and revenue generated by the viable topology currently used needs to be evaluated by comparing the continuously updated *cost and performance models* for each viable topology against the *Service Level Agreements (SLAs) and budget* associated with the service portfolio by the application owners. This approach requires, of course, that costs for the transition between viable topology models are negligible in comparison with the overall revenue generated by the application. Using a microservices-based approach for the decomposition of the service portfolio into isolated sub-systems before generating viable topologies would actually minimize such costs, since the finer granularity of each system tier would mean less components to consider (and potentially migrate) on the topology level. Alternatively, if this transition is deemed too costly and/or if the search space of viable topology alternatives has been exhausted then it is meaningful to revert to the previous stages of the lifecycle and either decompose the service portfolio as different α -topologies, or even refactor the service portfolio itself, repeating the cycle as necessary.

In order to add the necessary self-* mechanisms that regulate decision making during system operation a distributed MAPE-K model can be used [20], as discussed in Sect. 3. Considering the lifecycle of Fig. 1, however, it becomes clear that a hierarchical organization of controllers is better fitting. On the bottom level, it is possible to view the architectural components of each viable topology as its own autonomic element. However, all such elements need to coordinate with a controller on the level of the viable topology which is responsible for changes inside it. Another level of controllers is necessary to be added at the level of α -topologies when more than one viable topologies are active for a given decomposition. A similar process is repeated to the level of the service portfolio, and is used in order to trigger the transitions between stages of the lifecycle. Since the degree of automation that is feasible and available can vary among these transitions, it might become necessary to involve architects and system designers for this purpose. As such, design activities could be triggered by operations, as much as operational models could be derived during development.

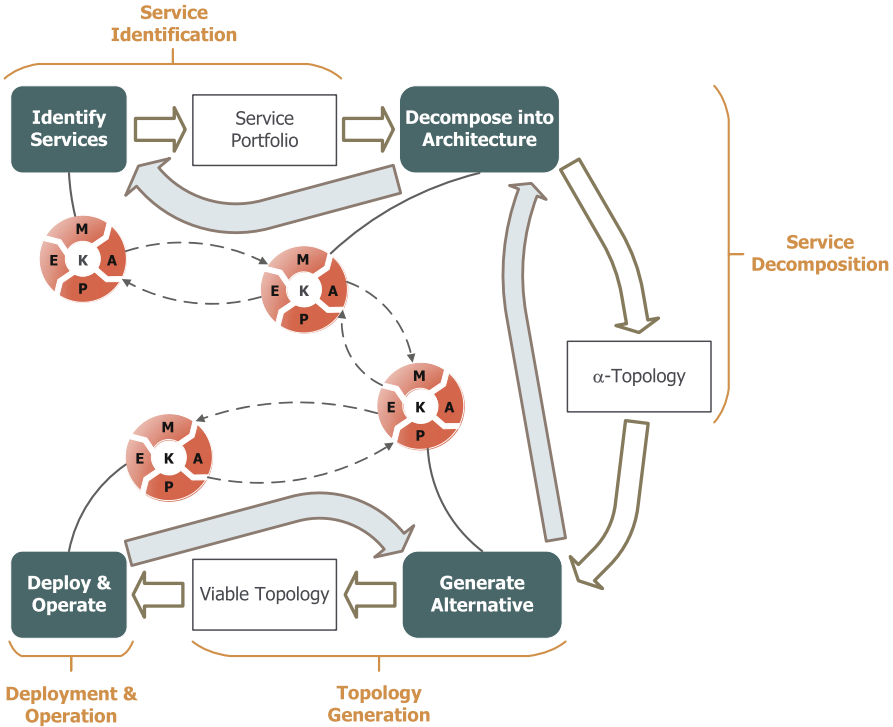


Fig. 2. The phases of the CBA lifecycle, with activities implemented as MAPE-K loops and information flowing between them

Figure 2 summarizes and illustrates this discussion by identifying the concrete *phases* of the proposed lifecycle (Service Identification, Service Decomposition, Topology Generation, and Deployment & Operation) and the *activities* that take place in each phase (Identify Services, Decompose into Architecture, Generate Alternative, and Deploy & Operate, respectively). Each of the activities in the figure is implemented by a MAPE-K controller which is responsible for monitoring the situation at its level (e.g. α -topology), analyzing its behavior (is the application within its SLA and budget constraints?), planning for an action if necessary (deciding whether to transition into a new viable topology by moving into the Generate Alternative phase, or into a new *alpha*-topology by escalating the decision upwards into the Service Identification controller), and executing the decided action. Rules for the decision making, and the outcomes of past decisions are persisted in the knowledge base component of the controller at each level in order to learn over time about the effectiveness of each decision in a given context. Figure 2 shows the flow of information between the controllers of each level as dashed arrows between the loops. Bottom-level controllers (i.e. the controllers of the components in a viable topology in the Deployment & Operation phase) can only decide to escalate the need for an adaptive action upwards in

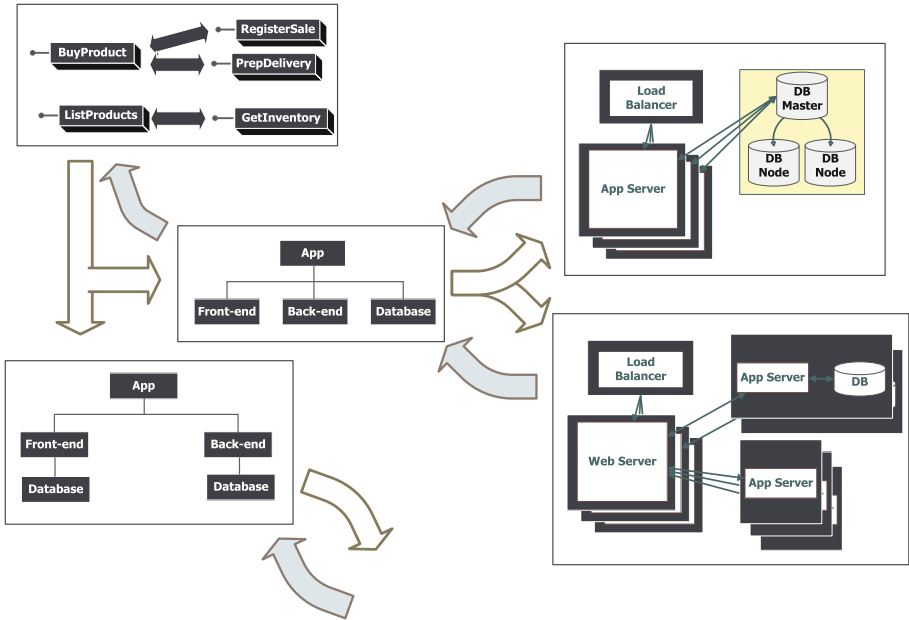


Fig. 3. The lifecycle of an example CBA (Web Shop)

the hierarchy, while top-level controllers (i.e. the controller at the level of Service Identification) can only trigger transitions into a lower level through the next phase.

4.2 An Example Instantiation

Figure 3 shows an example instantiation of the proposed lifecycle in the case of a Web Shop application. The service portfolio for the Web Shop consists (among others) of two client-facing services: BuyProduct and ListProducts, the former of which is composed out of services RegisterSale and PrepDelivery, while the latter one is using the internal service GetInventory. The BuyProduct service can be decomposed into a classic three-tier architecture, resulting in the top α -topology in the figure; for ListProducts a simpler two-tier architecture with separate (eventually synchronizing) databases is used. Staying with the first α -topology we can see that there are at least two alternative viable topologies to consider: in the first a DBaaS solution like AWS RDS¹⁰ is used for the Database tier, operating in a cluster mode for scalability purposes. The front- and back-end are implemented as a web application deployed inside an *App Server* like JBoss¹¹ that is scaled horizontally by running inside multiple VMs in a service like AWS EC2¹².

¹⁰ Amazon Relational Database Service: <https://aws.amazon.com/rds/>.

¹¹ JBoss: <http://www.jboss.org/>.

¹² Amazon EC2: <https://aws.amazon.com/ec2/>.

A Load Balancer solution is deployed inside its own VM for traffic routing. An alternative viable topology for the Web Shop consists of deploying the front-end in its own dedicated VM cluster, decoupling the stateless functionalities of the back-end and deploying them separately in their own stack, and bundling the rest of the back-end into VMs combining application servers and database instances that replace the DBaaS solution (but which still need some logic to synchronize). Such transformations require of course much more detailed α -topologies than the examples in Fig. 3 that are kept to a minimum for illustration purposes, but are nevertheless possible to be largely automated given an appropriate knowledge base of reusable software stacks expressed e.g. as γ -topologies [4].

4.3 Evaluation and Discussion

Looking at the requirements identified in Sect. 3, it can be seen that the proposed lifecycle indeed satisfies them by: (i) seamlessly integrating service-orientation concepts both at the level of the artifacts that it deals with (applications as service portfolios), and at the level of cloud services used as the underlying resources for the deployment and operation of an application; (ii) building around the dynamic nature of application topologies by decoupling their α -topology from the actual viable topology and relying on the generation of the latter on demand to cope with changes in the perceived behavior of the application through the hierarchy of MAPE-K controllers; (iii) implementing the foreseen self-* characteristics by means of the same controllers; and finally, (iv) by introducing awareness of the consumed resources across the different phases of the lifecycle. However, validation of the lifecycle in more complex scenarios than the example presented in the previous through e.g. field studies, is the subject of future work since it is related with the development of the necessary tooling to support it (see Sect. 6). Furthermore, and in terms of limitations to the presented work there are two main issues not covered by the discussion: quality assurance for the developed software, and security and privacy. Both of these issues are in practice cross-cutting concerns running in parallel to the lifecycle, and while it can be argued that they could therefore be considered external to it, they nevertheless need to be examined further in future works.

5 Related Work

There are a number of mature works in the literature focusing on the complete lifecycle of cloud-based applications that are related to the lifecycle proposed here. In their majority however they address only parts of the requirements discussed in Sect. 3. For example, the Cloud Application Lifecycle Model (CALM) and its supporting framework is introduced in [35] without a provision for self-* characteristics or cost awareness. The same holds for [26] that discusses a cloud application lifecycle from a service governance perspective, and for the lifecycle

presented in [28] which builds around the notion of blueprints as abstract templates for services to be published in application marketplaces. The work in [36] uses a centralized repository as the means to manage knowledge related to the phases of the lifecycle, but without the notion of cost awareness.

In further related work, the MODAClouds project relies on a Model-Driven Development-based approach to support the lifecycle of cloud-based applications [5]. The project builds on the `models@runtime` architectural pattern to connect run-time and design-time [13] and provides an IDE for the development, provisioning, deployment, and adaptation of CBAs. Nevertheless, the CBA lifecycle itself is only implicitly defined by this approach. The work in [7], part of the PaaSage project, discusses a service-based application lifecycle that emphasizes a multi-cloud deployment model. When compared to this work, the approach discussed in [7] attempts to (dynamically) optimize provider selection considering also monitoring data without however taking into account the possibility to re-distribute the application as part of this process.

6 Conclusions and Outlook

In summary, this work is based on the observation that the adoption of cloud computing, in conjunction with the advancements in software development in the form of DevOps, container-based software management, and microservices, requires an evolutionary step in software engineering practices, and especially in the area of software architecture. The challenges that drive this evolution are the everything as a service model in which cloud resources are offered, the multi-tenant environment created by resource pooling, the need to incorporate cost awareness due to the utility-based cost model for cloud computing, and the abundance of available offerings that can easily and efficiently replace parts of the software stack of each application. These challenges transform the lifecycle of cloud-based applications into a series of loops that transition between sets of application functionalities encapsulated as services, abstractly defined but application-specific architectural models, and software stack models that seamlessly incorporate cloud services. These transitions are triggered by controllers that coordinate within and across the various stages of the lifecycle.

Future work focuses on developing the methodologies and instrumentation necessary in order to support the proposed lifecycle, with a refinement of its various stages as an essential part of this process. A complete IDE in the manner discussed by the MODAClouds approach [5] is identified as the means to achieve this. Such an environment would further allow for field study-based validation of the lifecycle through collaboration with the industry. The development and integration of the IDE with the MAPE-K controllers as the implementation of the lifecycle phases-related activities is a critical component towards this effort.

References

1. Aceto, G., Botta, A., De Donato, W., Pescapè, A.: Cloud monitoring: a survey. *Comput. Netw.* **57**(9), 2093–2115 (2013)
2. Andrikopoulos, V., Binz, T., Leymann, F., Strauch, S.: How to adapt applications for the cloud environment. *Computing* **95**(6), 493–535 (2013)
3. Andrikopoulos, V., Bucchiarone, A., Di Nitto, E., Kazhamiakin, R., Lane, S., Mazza, V., Richardson, I.: Service engineering. In: Papazoglou, M.P., Pohl, K., Parkin, M., Metzger, A. (eds.) *Service Research Challenges and Solutions for the Future Internet*. LNCS, vol. 6500, pp. 271–337. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17599-2_8
4. Andrikopoulos, V., Gómez Sáez, S., Leymann, F., Wettinger, J.: Optimal distribution of applications in the cloud. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) *CAiSE 2014*. LNCS, vol. 8484, pp. 75–90. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_6
5. Ardagna, D., Di Nitto, E., Casale, G., Petcu, D., et al.: MODACLOUDS: a model-driven approach for the design and execution of applications on multiple clouds. In: *Proceedings of the 4th International Workshop on Modeling in Software Engineering, MiSE 2012*, pp. 50–56. IEEE Press, Piscataway (2012)
6. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., et al.: Above the clouds: a Berkeley view of cloud computing. Technical report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009. <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>
7. Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., et al.: Lifecycle management of service-based applications on multi-clouds: a research roadmap. In: *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, pp. 13–20. ACM (2013)
8. Bass, L., Weber, I., Zhu, L.: *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, Old Tappan (2015)
9. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the weather tomorrow?: towards a benchmark for the cloud. In: *Proceedings of the Second International Workshop on Testing Database Systems*, p. 9. ACM (2009)
10. Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q.Z., et al.: A service computing manifesto: the next 10 years. *Commun. ACM* **60**(4), 64–72 (2017)
11. Savage, M.: There’s just no getting around it: you’re building a distributed system. *Queue* **11**(4), 30 (2013)
12. Erl, T.: *SOA: Principles of Service Design*. Prentice Hall Press, Upper Saddle River (2007)
13. Ferry, N., Solberg, A.: Models@ runtime for continuous design and deployment. In: Di Nitto, E., Matthews, P., Petcu, D., Solberg, A. (eds.) *Model-Driven Development and Operation of Multi-Cloud Applications*, pp. 81–94. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46031-4_9
14. Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V., Tosun, C.: Benchmarking in the cloud: what it should, can, and cannot be. In: Nambiar, R., Poess, M. (eds.) *TPCTC 2012*. LNCS, vol. 7755, pp. 173–188. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36727-4_12
15. Heinrich, R., Schmieders, E., Jung, R., Rostami, K., et al.: Integrating run-time observations and design component models for cloud system analysis. In: *Proceedings of the 9th Workshop on Models@run.time*, vol. 1270, pp. 41–46. CEUR (2014)

16. Hüttermann, M.: Infrastructure as Code, pp. 135–156. Apress, Berkeley (2012)
17. Iosup, A., Yigitbasi, N., Epema, D.: On the performance variability of production cloud services. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 104–113. IEEE (2011)
18. Jula, A., Sundararajan, E., Othman, Z.: Cloud computing service composition: a systematic literature review. *Expert Syst. Appl.* **41**(8), 3809–3824 (2014)
19. Kashef, M.M., Altmann, J.: A cost model for hybrid clouds. In: Vanmechelen, K., Altmann, J., Rana, O.F. (eds.) *GECON 2011*. LNCS, vol. 7150, pp. 46–60. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28675-9_4
20. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. *Computer* **36**(1), 41–50 (2003)
21. Kratzke, N., Quint, P.C.: Understanding cloud-native applications after 10 years of cloud computing—a systematic mapping study. *J. Syst. Softw.* **126**, 1–16 (2017)
22. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: comparing public cloud providers. In: *Proceedings of the 10th Annual Conference on Internet Measurement, IMC 2010*, pp. 1–14. ACM, New York (2010). <http://doi.acm.org/10.1145/1879141.1879143>
23. Linthicum, D.S.: Moving to autonomous and self-migrating containers for cloud applications. *IEEE Cloud Comput.* **3**(6), 6–9 (2016)
24. Mell, P., Grance, T., et al.: The NIST definition of cloud computing. NIST Special Publication 800–145 (2011). <http://dx.doi.org/10.6028/NIST.SP.800-145>
25. Moldovan, D., Truong, H.L., Dustdar, S.: Cost-aware scalability of applications in public clouds. In: 2016 IEEE International Conference on Cloud Engineering (IC2E), pp. 79–88. IEEE (2016)
26. Munteanu, V.I., Fortis, T.F., Negru, V.: Service lifecycle in the cloud environment. In: 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 457–464. IEEE (2012)
27. Newman, S.: *Building microservices*. O’Reilly Media, Inc., Sebastopol (2015)
28. Nguyen, D.K., Lelli, F., Papazoglou, M.P., Van Den Heuvel, W.J.: Blueprinting approach in support of cloud computing. *Future Internet* **4**(1), 322–346 (2012)
29. Pahl, C., Jamshidi, P.: Software architecture for the cloud – a roadmap towards control-theoretic, model-based cloud architecture. In: Weyns, D., Mirandola, R., Crnkovic, I. (eds.) *ECSCA 2015*. LNCS, vol. 9278, pp. 212–220. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23727-5_17
30. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: state of the art and research challenges. *Computer* **40**(11), 38–45 (2007)
31. RightScale: RightScale 2017 State of the Cloud Report (2017). <https://www.rightscale.com/lp/state-of-the-cloud>
32. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow.* **3**(1–2), 460–471 (2010)
33. van Steen, M., Tanenbaum, A.S.: A brief introduction to distributed systems. *Computing* **98**(10), 967–1009 (2016)
34. Sun, L., Dong, H., Hussain, F.K., Hussain, O.K., Chang, E.: Cloud service selection: state-of-the-art and future research directions. *J. Netw. Comput. Appl.* **45**, 134–150 (2014)
35. Tang, K., Zhang, J.M., Feng, C.H.: Application centric lifecycle framework in cloud. In: 2011 IEEE 8th International Conference on e-Business Engineering (ICEBE), pp. 329–334. IEEE (2011)

36. Tran, H.T., Feuerlicht, G.: Service repository for cloud service consumer life cycle management. In: Dustdar, S., Leymann, F., Villari, M. (eds.) ESOCC 2015. LNCS, vol. 9306, pp. 171–180. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24072-5_12
37. Walker, E.: The real cost of a CPU hour. *Computer* **42**(4), 35–41 (2009)
38. Weinman, J.: Clouonomics: a rigorous approach to cloud benefit quantification. *J. Softw. Technol.* **14**(4), 10–18 (2011)
39. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. *J. Internet Serv. Appl.* **1**(1), 7–18 (2010)



A Cloud Computing Workflow for Managing Oceanographic Data

Salma Allam¹, Antonino Galletta²(✉), Lorenzo Carnevale², Moulay Ali Bekri¹, Rachid El Ouahbi¹, and Massimo Villari²

¹ Lab MIASH and Lab MACS, Department of Computer Science and Mathematics, Faculty of Sciences, University Moulay Ismail, Meknes, Morocco

allam.salma@gmail.com, ali.bekri@gmail.com, elouahbi@yahoo.fr

² Department of Engineering, University of Messina, Messina, Italy

{angalletta, lcarnevale, mvillari}@unime.it

Abstract. Ocean data management plays an important role in the oceanographic problems, such as ocean acidification. These data, having different physical, biological and chemical nature, are collected from all seas and oceans of the world, generating an international networks for standardizing data formats and facilitating global databases exchange. Cloud computing is therefore the best candidate for oceanographic data migration on a distributed and scalable platform, able to help researchers for performing future predictive analysis. In this paper, we propose a new Cloud based workflow solution for storing oceanographic data and ensuring a good user experience about the geographical data visualization. Experiments prove the goodness of the proposed system in terms of performance.

Keywords: Oceanography · Cloud Computing · Data collection
Data management · Data migration · NoSQL · Big Data

1 Introduction

Ocean Data management is a current challenge because both of ocean specific terminology diversity (physio-chemical parameters, sensor type, units of measures, conditions of measures, etc.) and of huge volume of ocean data collected from several international projects. The last aim to control the ocean acidification phenomena, an emerging global problem related to the seawater CO₂ rate [1] that negatively affects the environment. Therefore, scientific community was thinking about software for calculating inorganic seawater carbon in order to track the evolution of CO₂ in the oceans.

However, traditional desktop or web application can not provide the functionalities required by similar problem. Indeed, storing and processing a big volumes of data needs availability, reliability and scalability. For this purpose, the best choice for this kind of application is Cloud Computing, which delivers the resources for managing efficiently the collected data.

The goal of this scientific work follows the previous one [2]. More specifically, in this paper we planned to improve scalability and user experience of the Web Application, used for visualizing oceanographic data, already developed. For this purpose, here we analyzed the oceanographic data contest in order to design a Cloud workflow for migrating data from online databases to a distributed system able to enable future predictive analysis. Thus, we designed a data acquisition and integration workflow through a Cloud Storage approach which use a more recommended NoSQL solution for successfully managing semi-structured data and retrieving them for future seawater’s acidification predictive analysis.

The rest of the paper is organized as follows. Related Works are described in the Sect. 2. In Sect. 3, we discussed the material used in this scientific work, from data source up to main oceanographic data issues. The Sect. 4 explains the Cloud approach used in order to migrate oceanographic data from sources to Cloud Storage, whereas in Sect. 5 we discussed the outcomes’ experiments. Finally, the Sect. 6 concludes the paper with the lights for the future.

2 Related Work

The most popular oceanographic data visualization software is the Ocean Data View (ODV). According to Schlitzer [3], ODV is a software used for the interactive exploration, analysis and visualization of oceanographic and other georeferenced profile, time-series, and trajectory or sequence data. It displays original data points or gridded fields based on the original data and supports different data formats. ODV displays data on different views representing it in a global map that integrates the gridding algorithms [4] based software, called DIVA [5], in order to grid elements in the map for performing interpolation. Moreover, it allows to select one data source by entering the outer coordinates, considering the result as a separate small collection. In addition, ODV allows to select features for drawing one or more specific diagrams in order to compare these.

A software for 3D visualization has been proposed by Ware et al. [6]. The representation of that requires a user visual stimulation and allows them to compare two or more locations [7].

On the other hand, in recent time, the scientific community has started an investigation about atmospheric and oceanographic research using the Cloud Computing paradigm [8]. In order to proof that, in [9], the author reported a survey for discussing the progress made by Cloud in the oceanographic challenges. These include effective discovery, organization, analysis and visualization of large amounts of data. In [10], the authors reported “*the outcomes of an NSF-funded project that developed a geospatial cyberinfrastructure to support atmospheric research*”. Specifically, they provided several modules for covering the aforementioned challenges in order to “*develop an online, collaborative scientific analysis system for atmospheric science*”. In [11], instead, the authors described the LiveOcean project, which aims to mitigate “*the financial impact of ocean acidification on the shellfish industry in the Pacific Northwest of the United States*”. The authors builded this system on Microsoft Azure Cloud Platform highlighting the modularity as most important theme.

Other important aspect is the management of the sensors designated for gathering raw data. Indeed, increasing the number of data also the oceanographic context entries in the Big Data problem and specific solutions, such as [12, 13], are useful for be inspired.

Our approach aims to go over the [3–7] solutions, proposing a Cloud Computing scenario in order to provide for Big Data coming into the oceanographic context.

Cloud Computing is a very hot topic in scientific community, it raises challenges in research fields as described in [14–17]. Most of the works are focused on the study and realization of innovative models that allows the collaboration among different Cloud providers focusing on various aspects of the federation. New trends in scientific work aim to adopt the Federation for new interesting scenarios: IoT, Edge, Cloud and Osmotic Computing [18].

3 Material

The following section describes the material used for this scientific work, highlighting the data structure and discussing the main challenges and issues.

3.1 Data Source

Data used in this scientific work came from the Carbon Dioxide Information Analysis Center (CDIAC), which is considered the first information analysis center for the oceanic parameters [19]. It provides an important climate-change database center organized as showed in the Fig. 1.

In particular, with reference to the Fig. 1, it is possible to notice the ODVServer zone, that represents the data sources selected in our study. Data are stored into relational databases, it is possible to export them using the comma separated value (CSV) format. In particular the ODVServer Data System is composed by three databases:

1. the GLObal Ocean Data Analysis Project (GLODAP) which gathers unified dataset for determining the “*global distributions of both natural and anthropogenic inorganic carbon, such as radiocarbon*” [20];
2. the PACIFIC ocean Interior CARbon (PACIFICA) database that gathers “*data synthesis of ocean interior carbon and its related parameters in the Pacific Ocean*” [21];
3. the CARbon dioxide IN the Atlantic Ocean (CARINA) database which gathers “*data set of open ocean subsurface measurements for biogeochemical investigations*” [22].

Other data sources (SOCAT, CORILIOS CORA, JGOFS, eWOCE, LDEO and CLIVAR) are out of the scope of this paper and will be treated in future works.

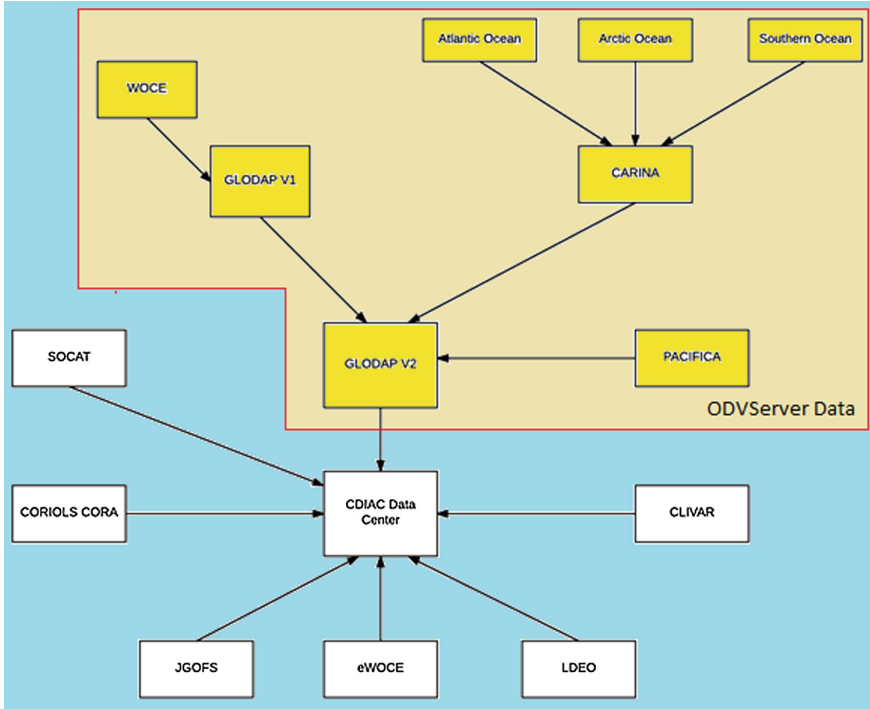


Fig. 1. CDIAC data center

3.2 Data Structure

Data collected in the aforementioned databases have a common structure explained in the following:

- **Time data:** Month, Day, Year;
- **Location data:** Longitude, Latitude, Depth;
- **Physical & Chemical data:** Section, Station, Cruise, BottomDepth, BottleNumber, Cast, Salinity, cdtSalinity, Oxygen, Nitrate, Nitrite, Silicate, Phosphate, CFC11, CFC12, CFC113, TCO2, Alkalinity, pCO2, pHSWS25, pHSWS25_Temp, AnthropogenicCO2, DOC, TOC, DeltaC14, DeltaC13, H3, DeltaH3, He, C14err, H3err, DeltaH3err, He_err, CC14, SF6, AOU, pCFC11, CFC11Age, pCFC12, pCFC113, pCCI4, pSF6, CFC12Age, PotentialAlkalinity, ConventionalRadiocarbonAge, NaturalC14, bkgc14e, BombC14, BombC14atom, NaturalC14atom, PotTemperature, SigmaTheta, Sigma1, Sigma2, Sigma3, Sigma4, bf, sf, cdtsf, of, no3f, no2f, sif, po4f, cfc11f, cfc12f, cfc113f, tco2f, alkf, pco2f, phsWS25f, aco2f, docf, tocf, c14f, c13f, h3f, dh3f, hef, ccl4f, sf6f, aouf, palkf, bkgc14f, bombc14f;
- **Environmental data:** Pressure, Temperature.

Data are collected according to the specific oceanographic region. Specifically, each region is composed by a set of sections that contain several stations

geographically located. For each of them, there are more than one record depending on the depth variation. The Fig. 2 represents a hierarchical overview of the entire dataset just described.

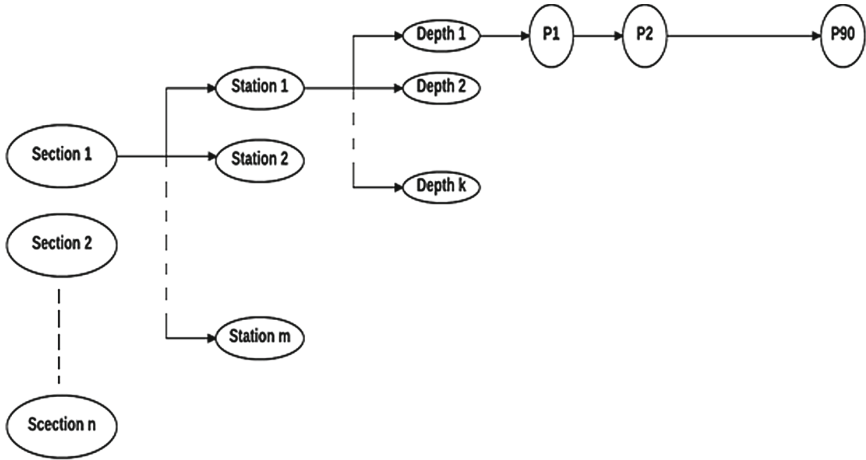


Fig. 2. Hierarchical multivalued attributes. Here P1, P2, ... , P90 denote Physical & chemical data.

3.3 Challenges and Issues of Oceanographic Data

The CDIAC oceanographic data have the ‘Cruise/Section-Station-Depth’ form. Such data are in relationships with time and space and archive all information about how, when and where data were stored, as well as type and nature of available data. Thus, it provides a conceptual overview of ocean data structure that should be useful in data management.

In ocean data, the dimensions are normally recorded as ‘date/time, latitude, longitude, and depth’ [23]. By associating latitude and longitude in location, the model will be ‘date/time, location, and depth’. Unfortunately, data provided by the GLODAP, PACIFICA and CARINA databases do not have the data/time field in the ‘dd/mm/yy hour:minute:second’ form, but only in the ‘dd/mm/yy’ form. Therefore, the model becomes ‘data, location, depth’. For this purpose, it is difficult to treat these data as time series.

Referring to the TLZ model [23], there are eight possible relationships, as reported in the Table 1.

We also noted that the geographical distance and depth gauge is not periodic among samples, i.e the Fig. 3 shows that the distance among stations is not the same and the stations depth is also different. More specifically, the data gauge on different depth is not uniform throughout the sample collection process.

Thus, the main challenge was to manage these data, in order to facilitate future predictive analysis and query them in a flexible way.

Table 1. DLH space relationship. Noted Date = ‘D’, Location = ‘L’ and Depth = ‘H’.

| D L H | Relationship |
|-------|--------------------------------------|
| 1 1 1 | One date, one location, one depth |
| 1 1 n | One date, one location, many depth |
| 1 n 1 | One date, many location, one depth |
| 1 n n | One date, many location, many depth |
| n 1 1 | many date, one location, one depth |
| n 1 n | many date, one location, many depth |
| n n 1 | many date, many location, one depth |
| n n n | many date, many location, many depth |

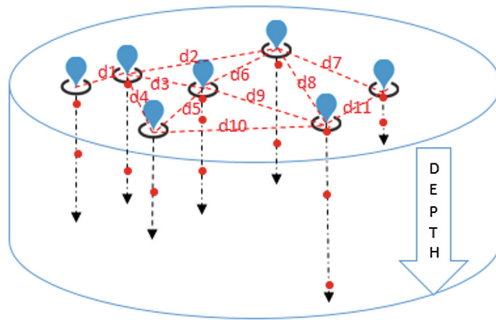


Fig. 3. A representation of data characteristic

4 Methodology

As mentioned in the Sect. 1, this work aimed to improve the previous one [2] through the Cloud Computing utilization. Specifically, the Sun/Oracle’s JEE-based cross-platform, called ODVServer, used to store data in a traditional SQL database and to visualize oceanographic data using Google Maps APIs. The Fig. 4 shows an overview of the previous platform.

Unfortunately, the visualization of all sections from one database was not easily distinguishable. Moreover, we were not able to analyze the oceanographic data on the basis of different geographical shapes. For this purpose, we propose the Cloud improvement reported in the following.

4.1 Workflow

Driven by the need to improve the viewing system of the different oceanographic sections, we have planned to replace web data processing, performed with the Google Maps APIs, with the native storage of a GeoJSON, a human and machine-readable format for encoding geographic data structures.

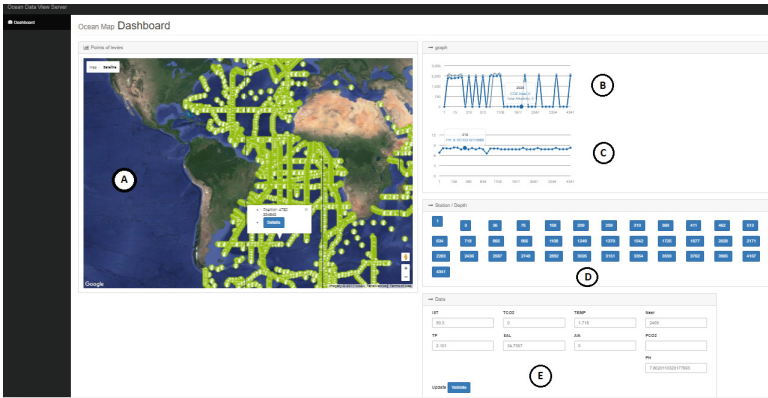


Fig. 4. Overview of the ODVServer platform. The layout was divided into two side. The first one (A) provided a geographical representation of the reading. Indeed, the right side (B, C, D, E) provided the insight view of the selected station.

Therefore, a NoSQL database was required in order to manage semi-structured and non structured data and for storing all the oceanographic databases.

Referring to the Fig. 5, data move from sources (CARINA, PACIFICA and GLODAP) to the Cloud Platform through RESTful APIs. Specifically, the listening microservice receives CSV data in order to implement the transformation into GeoJSON. Thus, this information moves to a sharded and replicated MongoDB distribution, a native JSON NoSQL database. The choice of MongoDB avoids further data transformations. Moreover, in order to scale the microservice workload, it is embedded inside a docker container, ensuring a lightweight and portable service virtualization.

The bottom side of the Fig. 5 shows a HTTP communication between the distributed storage and the front end, in order to view the query and future analyses results. At the same time, Apache Spark has been thought for performing future real-time predictive algorithms on oceanographic data. However, the dotted lines indicate guidelines for future works.

4.2 Oceanographic Data Visualization

Based on the previous description, we looked for two properties: interoperability and flexibility. The first one is guaranteed by the GeoJSON standard. Its fixed structure identifies two parts: **geometry** section contains geospatial information and type of GeoJSON element (see Sect. 2 in the Fig. 6); and **properties** section contains all parameters codified as key-value pairs (see Sect. 3 in the Fig. 6). We remark that Sect. 1 in the Fig. 6 represents the MongoDB's master key.

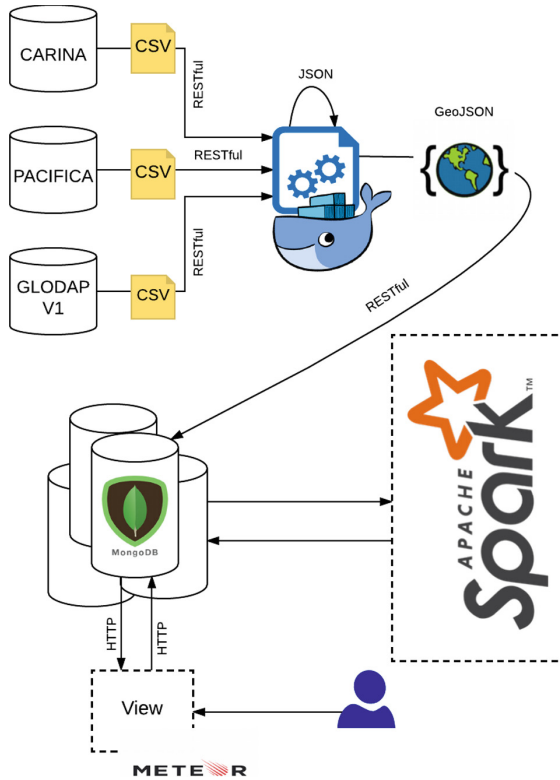


Fig. 5. The workflow includes the data acquisition and integration phases, considering the CSV format databases (CARINA; PACIFICA and GLODAP). A NoSQL solution stores all the GeoJSON information and integrates well with data analysis tool, such as Apache Spark, and MEAN stack web application, such as the Meteor JS framework. The dotted lines indicates guidelines for future works.

On the other hand, the flexibility is guaranteed by data management and visualization dynamism, which allow users to select any representation. Indeed, in our approach, we decided to store all samples in a single MongoDB collection. Thus, we can create virtual representation based on users' demand. For instance, as showed in the Fig. 7, by means of our approach users are able to create virtual representations per each zone of interest, starting from position of samples or other constrains.

```
1 {
  "id": "ObjectId("592463bd56b8e78817719be8")",
  "Section": "06GA19960613",
  "Station": 6,
  "Cruise": "06GA19960613",
  "Longitude": 30.5833,
  "Latitude": 81.205,
  "Month": 8,
  "Day": 12,
  "Year": 1993,
  "Pressure": 174.7,
  "Depth": 173,
  "Temperature": 1.3228,
  "Salinity": 34.7401,
  "Oxygen": 312.012,
  "Silicate": 5.44,
  "TCO2": 2125.8,
  "Alkalinity": -999,
  "pCO2": -999,
  "AnthropogenicCO2": -999,
  "sf": 2,
  "of": 2,
  "sif": 2,
  "tco2f": 2,
  "alkf": 9,
  "pco2f": 9,
  "aco2f": 9
}

1 {
  "id": "ObjectId("5935834cafb35918eb5f4675")",
  "geometry": {
    "coordinates": [
      28.069,
      -33.2493,
      2.0
    ],
    "type": "Point"
  },
  "type": "Feature",
  "properties": {
    "Section": "WOCE_I06Sb",
    "Station": 2,
    "Cruise": "35MF103_1",
    "Longitude": 28.069,
    "Latitude": -33.2493,
    "Month": 2,
    "Day": 21,
    "Year": 1996,
    "Pressure": 2.3,
    "BottomDepth": 628,
    "BottleNumber": 19,
    "Cast": 1,
    "Depth": 2,
    "Temperature": 25.3721,
    "Salinity": 35.2756,
    "cdtSalinity": -999
  }
}
2
3
```

JSON format

GeoJSON format

Fig. 6. Difference structure of JSON and GeoJSON

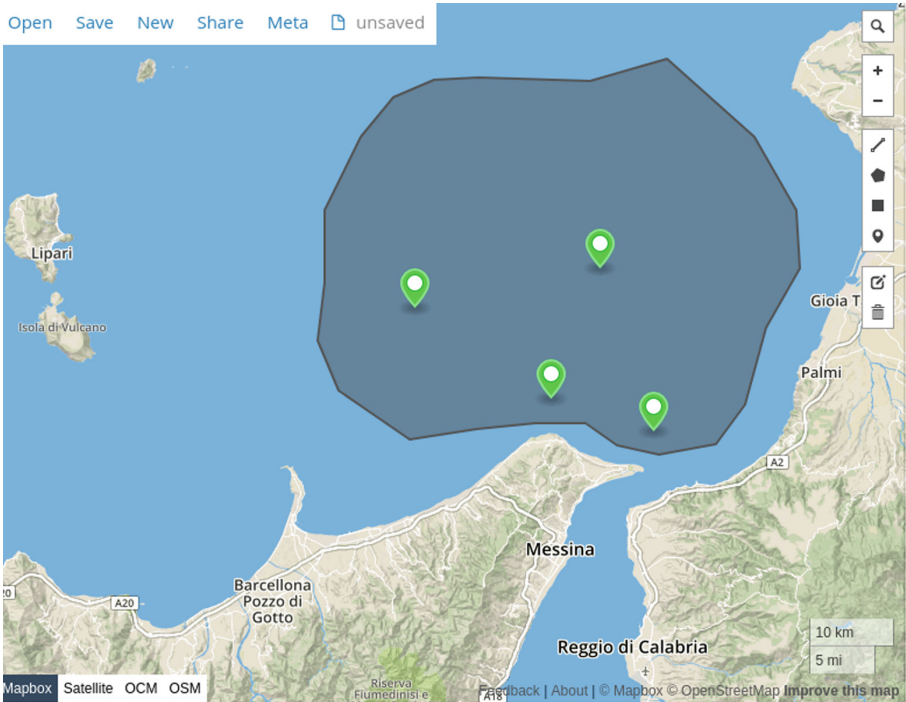


Fig. 7. User demand

5 Performance

In this section we discuss about of the performances of the system from a numerical point of view. In particular, we conducted two different kind of analysis: for populating and retrieve data of our system. Our testbed is composed of two different blades server. More specifically, we have 2 different type of machines one for the computation and the other one for the storage. Computation workstation, in which is running the data conversion module, is equipped by the Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz, RAM 16 GB, OS: Ubuntu server 16.04 LTS 64 BIT. Storage workstation, in which our database system MongoDB is running on single node, is composed by the Intel(R) Core(TM) i3-6100 CPU @ 3.70 GHz, RAM 32 GB, and Ubuntu server 16.04 LTS 64 BIT. Unfortunately we did not find any solution to compare performances of our system. We made scalability tests in different scenarios for both type of analysis. Experiments were repeated 30 subsequent times in order to consider confidence intervals at 95% and average values.

5.1 Insert Data

Figure 8 shows the performances for parsing CSV data to GeoJSON and storing them into MongoDB. Its behavior is linear with the increasing of the dataset size. On the x-axis we reported the dataset size, whereas on the y-axis, we reported the response time expressed in msec. How we can observe, the response time for 100.000 samples is acceptable less than 10 s.

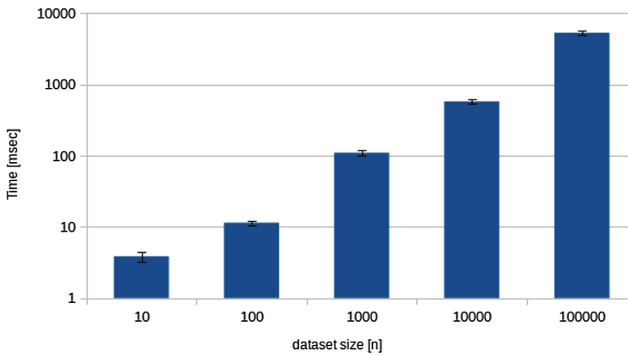


Fig. 8. Data insert performance

5.2 Retrieve Data

Here we consider times for retrieving data from MongoDB in a specific geographic shapes, in order to understand if flexibility features are really implementable. More specifically we considered increasing concentric circles with different radius, starting from 10 m up to 100 km.

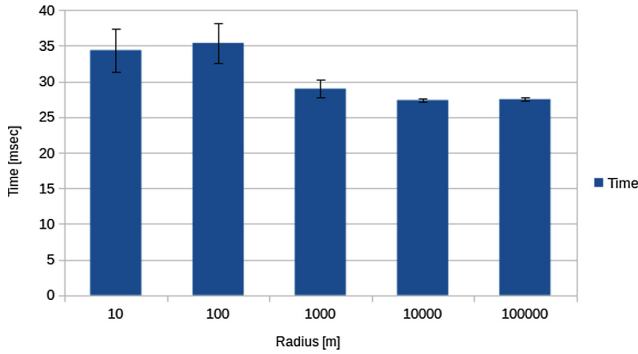


Fig. 9. Data retrieve performance

The behavior, as showed in Fig. 9, is constant around 30 ms, variations are due to networks delay.

6 Conclusions and Future Work

In this scientific work, we investigated the management of oceanographic data through the utilization of a Cloud Computing workflow. First of all, three CSV format databases have been selected as data sources. Therefore, we explained the workflow necessary for migrating these data up to the Cloud Storage. This scientific work is the first initiative adopting Cloud for manage Ocean data, for this reason we did not find any solution to compare performance of our system. However experiments showed that our system response time presents a linear trend. The execution time grows up with the increasing number of considered samples.

On the other hand, the dotted lines in the Fig. 5 shows our idea about the future perspective. In particular, Meteor JS framework will be the technology we

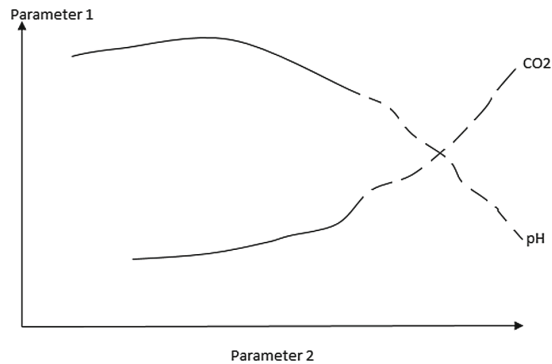


Fig. 10. A goal of the future work can be the acidification prediction.

aim to use for developing the new frontend version. This choice depends on the native MEAN stack adopted, which includes MongoDB as backend database; whereas Apache Spark will be useful for performing predictive oceanographic data analysis, such as the acidification prediction. About that, the Fig. 10 shows a possible future work about the selection of the features that best describe the reported behavior. Other future works are related to adopt the new Osmotic Computing paradigm.

Acknowledgment. This work has been supported by Cloud for Europe grant agreement number FP7-610650 (C4E) Tender: *REALIZATION OF A RESEARCH AND DEVELOPMENT PROJECT (PRE-COMMERCIAL PROCUREMENT) ON “CLOUD FOR EUROPE”*, Italy-Rome: Research and development services and related consultancy services Contract notice: 2014/S 241-424518. Directive: 2004/18/EC. (<http://www.cloudforeurope.eu/>).

References

1. Doney, S.C., Balch, W.M., Fabry, V.J., Feely, R.A.: Ocean acidification a critical emerging problem. *Oceanography* **22**(4), 16–25 (2009)
2. Allam, R.E.S., Ouahbi, M.D.E.O.: *Adv. Inf. Technol. Theory Appl.* **1**, 163–166 (2016). ISSN: 2489–1703
3. Schlitzer, R.: *Ocean Data View*, pp. 1–11 (2011)
4. Smith, W.H.F., Wessel, P.: Gridding with continuous curvature splines in tension. *Geophysics* **55**(3), 293–305 (1990). <http://library.seg.org/doi/10.1190/1.1442837>
5. Started, G.: *Ocean Data View*, pp. 1–11 (2011)
6. Ware, C., Plumlee, M., Arsenault, R., Mayer, L.A., Smith, S., House, D.: GeoZui3D: data fusion for interpreting oceanographic data. In: *Oceans Conference Record (IEEE)*, vol. 3, pp. 1960–1964 (2001)
7. Plumlee, M., Ware, C.: An evaluation of methods for linking 3D views. In: *Proceedings of the Symposium on Interactive 3D Graphics*, pp. 193–201 (2003). <http://www.scopus.com/inward/record.url?eid=2-s2.0-0038642661&partnerID=tZOtx3y1>
8. Butler, K., Merati, N.: Analysis patterns for cloud-centric atmospheric and ocean research. In: *Cloud Computing in Ocean and Atmospheric Sciences*, pp. 15–34. Elsevier (2016). <https://doi.org/10.1016/b978-0-12-803192-6.00002-5>
9. Wigton, R.: Forces and patterns in the scientific cloud. In: *Cloud Computing in Ocean and Atmospheric Sciences*, pp. 35–41. Elsevier (2016). <https://doi.org/10.1016/b978-0-12-803192-6.00003-7>
10. Li, W., Shao, H., Wang, S., Zhou, X., Wu, S.: A2ci. In: *Cloud Computing in Ocean and Atmospheric Sciences*, pp. 137–161. Elsevier (2016). <https://doi.org/10.1016/b978-0-12-803192-6.00009-8>
11. Fatland, R., MacCready, P., Oscar, N.: LiveOcean. In: *Cloud Computing in Ocean and Atmospheric Sciences*, pp. 277–296. Elsevier (2016). <https://doi.org/10.1016/b978-0-12-803192-6.00014-1>
12. Fazio, M., Celesti, A., Villari, M., Puliafito, A.: The need of a hybrid storage approach for IoT in PaaS cloud federation. In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 779–784 (2014)

13. Celesti, A., Peditto, N., Verboso, F., Villari, M., Puliafito, A., Draco PaaS: a distributed resilient adaptable cloud oriented platform. In: IEEE International Symposium on Parallel Distributed Processing. Workshops and PhD Forum 2013, pp. 1490–1497 (2013)
14. Tusa, F., Celesti, A., Villari, M., Puliafito, A.: How to enhance cloud architectures to enable cross-federation. In: Proceedings of IEEE CLOUD 2010, pp. 337–345. IEEE, July 2010
15. Vernik, G., Shulman-Peleg, A., Dippl, S., Formisano, C., Jaeger, M., Kolodner, E., Villari, M.: Data on-boarding in federated storage clouds. In: 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD), pp. 244–251, June 2013
16. Goiri, I., Guitart, J., Torres, J.: Characterizing cloud federation for enhancing providers' profit. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 123–130, July 2010
17. Azodolmolky, S., Wieder, P., Yahyapour, R.: Cloud computing networking: challenges and opportunities for innovations. IEEE Commun. Mag. **51**(7), 54–62 (2013)
18. Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R.: Osmotic computing: a new paradigm for edge/cloud integration. IEEE Cloud Comput. **3**(6), 76–83 (2016)
19. Catalog of Databases and Reports, May 1999. <http://cdiac.ornl.gov/oceans/>
20. Key, R., Olsen, A., van Heuven, S., Lauvset, S., Velo, A., Lin, X., Schirnack, C., Kozyr, A., Tanhua, T., Hoppema, M., Jutterström, S., Steinfeldt, R., Jeansson, E., Ishi, M., Perez, F., Suzuki, T.: Global Ocean Data Analysis Project, Version 2 (GLODAPv2). Ornl/Cdiac-162, Ndp-093, vol. 2 (2015)
21. Suzuki, T., Ishii, M., Aoyama, M., Christian, J.R., Enyo, K., Kawano, T., Key, R.M., Kosugi, N., Kozyr, A., Miller, L.A., Murata, A., Nakano, T., Ono, T., Saino, T., Sasaki, K.-I., Sasano, D., Takatani, Y., Wakita, M., Sabine, C.L.: Pacifica Data Synthesis Project. Ornl/Cdiac-159, Ndp-092 (2013)
22. Hoppema, M., Velo, A., van Heuven, S., Tanhua, T., Key, R.M., Lin, X., Bakker, D.C.E., Perez, F.F., Ríos, A.F., Lo Monaco, C., Sabine, C.L., Álvarez, M., Bellerby, R.G.J.: Consistency of cruise data of the CARINA database in the Atlantic sector of the Southern Ocean. Earth Syst. Sci. Data **1**, 63–75 (2009)
23. Hubbs, C.L.: University of Michigan, U.S.A., vol. III, pp. 1–6 (1930)



An Ontology-Based Architecture for an Adaptable Cloud Storage Broker

Divyaa Manimaran Elango¹, Frank Fowley¹, and Claus Pahl²(✉)

¹ IC4, Dublin City University, Dublin, Ireland

² Software and Systems Engineering Research Centre,
Free University of Bozen-Bolzano, Bolzano, Italy
Claus.Pahl@unibz.it

Abstract. Interoperability and easier migration between offered services are aims that can be supported by cloud service brokerage in the cloud service ecosystem. We present here a multi-cloud storage broker, implemented as an API. This API allows objects and collections of objects to be stored and retrieved uniformly across a range of cloud-based storage providers. This in turn realizes improved portability and easy migration of software systems between providers and services.

Our multi-cloud storage abstraction is implemented as a Java-based multi-cloud storage API and supports a range of storage providers including GoogleDrive, DropBox, Microsoft Azure and Amazon Web Services as sample service providers. We focus on the architectural aspects of the broker in this paper. The abstraction provided by the broker is based on a layered ontological framework. While many multi-cloud applications exist, we investigate in more detail the mapping of the layered ontology onto a design pattern-based organisation of the architecture. This software architecture perspective allows us to show how this satisfies important maintainability and extensibility properties for any software system.

Keywords: Cloud Service Brokerage · Cloud storage
Data migration · Ontology · API performance

1 Introduction

Interoperability is a key concern in the cloud service ecosystem. Cloud service brokerage (CSB) aims for more interoperability to enable more portability and easier migration between different service providers [26, 33, 34]. CSBs can support portability and migration through mechanisms such as integration and adaptation of different provided services into a uniform representation [15].

We present here a multi-cloud storage broker that implements an API to allow objects to be stored and retrieved uniformly across a range of storage providers. Two features characterise the broker. Firstly, the abstraction is based on a layered ontological framework that allows mapping of common concepts of object storage to implementation layers. Secondly, the architecture is organised

around common software design patterns to ensure maintainability and extensibility. This is important in order to extend the broker to new providers [21].

The central software architecture concepts for the design of the broker and methodology behind the abstraction library will be our core focus. The multi-cloud storage abstraction is realized by a Java-based multi-cloud storage API. Technically, the library is provided as a jar file that supports the selected four service providers, namely GoogleDrive, DropBox, Microsoft Azure and Amazon Web Services [37–40]. The library offers three service categories that reflect the different storagetypes, i.e., a file service, a blob service and a table service.

We focus on the ontology framework for the central storage concepts and functions and show how this is mapped onto a layered, design pattern-based library architecture for the API we developed [30,31]. This is an aspect that has not been sufficiently address in other investigations of multi-cloud brokers. An application of the library can also be used to compare storage operations across different providers, which we and others [16,32] have explored elsewhere.

This document is organized as follows. In Sect. 2, we give an outline of cloud service brokerage. Section 3 describes background and related work. In Sect. 4, the ontology-based interoperability framework is explained, and Sect. 5 looks at other architectural design aspects. Section 6 discusses the implementation effort and the learning outcome and presents some conclusions.

2 Principles of Cloud Service Brokerage and Use Cases

2.1 Cloud Brokerage

A cloud broker [11,17] is an intermediary software application between a client and cloud provider service. Brokerage reduces the time spent by a client in analyzing different types of services provided by different service providers.

In our case, brokerage enables a single platform to offer the client a common cloud storage service. This results in cost optimization and reduced level of back-end data management requirements, but also enables easy migration of data and files through the joint interface [8].

A multi-cloud storage abstraction API can act as the cloud broker library which facilitates the integration of different types of cloud services [18]. The abstraction library allows the broker to adapt to a rapidly changing marketplace [4]. Changeability and extensibility are consequently central requirements for our broker library [9,10]. Figure 1 illustrates the architecture.

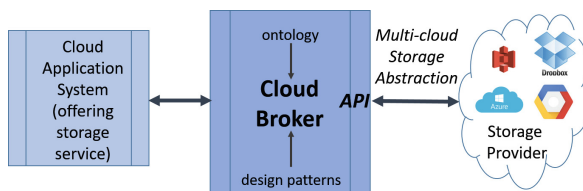


Fig. 1. Service brokerage architecture for cloud storage.

2.2 A Brokerage Use Case

Cloud brokerage shall be illustrated by a use case. Disaster recovery (DR) is a sample specific storage use case, used where there is an interruption of an action or an event in an unpredictable time that causes the services to be unavailable to the end user. Cloud back-up storage is a way of protecting the online resources to make them available in the event of a disaster, such as loss of data.

Our storage abstraction library is suited to support this DR use case as it provides a multi-cloud broker for easy storage back up. Concrete advantages are good time management in terms of restoring processes, increased scalability, security and compliance, redundancy and end to end recovery for the DR application [2,3]. The storage providers supported by our API (Microsoft Azure, Google, DropBox and Amazon Web Service) offer good bandwidth and low cost services that can be used for backup and recovery tasks.

2.3 Vendor Lock-In

Vendor lock-in is a problem in cloud computing, preventing users from migrating between providers. Clients become dependent on a single cloud provider. The client is not given an option to migrate to other providers. Issues can arise, such as legal constraints or increased costs, that consequently negatively impact on key properties by vendor lock-in and lack of standards [6,7].

A multi-cloud storage API can play a crucial role in such cases, making it easier for the client to switch providers. This can be applied across different cloud type environments, like private or public environments which are more beneficial from a business perspective. Furthermore, the extensibility of the library to support new cloud providers gives the client a wide view of portability to many different new cloud providers.

3 Background and Related Work

3.1 Cloud Service Provider APIs

Many cloud storage provider APIs exist, from which we selected four providers with different individual services [37–40]. Some key properties from a software engineering perspective that have impacted on the implementation are:

- Amazon Web Service S3: is a file storage service built on REST and SOAP. An S3 SDK is available in major development languages. The developer portal includes rich documentation. However, from a software development point-of-view, the services have a high number of classes. The library is heavy since it has many packages for all services. Understanding the class naming can be seen as challenging – many services are listed in the same SDK documentation. We also experienced the service be inconsistent, as there was an occasional delay in read and write requests.

- Azure Storage: supports blob, file, queue and table services. The API is built on REST, HTTP and XML, and can be easily integrated with Microsoft Visual Studio, Eclipse and GIT. Azure is relatively user friendly. The standard portal interface is used for storage account set-up and document DB account parameters. The Azure SDK is available for major development languages. The Azure SDK provides a separate API package for each service and has the same code flow across different service APIs.
- DropBox: is a file hosting service. It uses SSL transfer for synchronization and AES 256 encryption as the security mechanisms. It also enables synchronised backup and web sharing. The DropBox API is lightweight and easy for a new user to go through quickly. Code samples and method explanations are given in the developer’s portal.
- GoogleDrive: offers a cloud file storage service. The API is built on OAuth2 authentication. It is generally easy to understand. The structure is clearly documented and the use of method calls is well explained. The GoogleDrive service includes access to a Google API client library. Failure to include http and OAuth client libraries will disable the authentication. The Google developer portal simplifies the way of implementing the API in a workspace and provides details for configuring the authentication.

A survey of the main features of the providers that we carried out has resulted in a grouping of the cloud providers and their services as shown in Table 1.

Table 1. Storage services and their providers.

| Service | Azure | AWS | Google | DropBox |
|---------|--|-------------------------------------|-------------|---------|
| File | Azure storage file | - | GoogleDrive | DropBox |
| Blob | Azure storage blob | AWS S3 | - | - |
| Table | Azure storage table, Azure DocumentDB | Amazon DynamoDB, Amazon SimpleDB | - | - |

3.2 Multi-cloud Libraries

For the design of our multi-cloud broker, we looked at existing multi-cloud libraries for inspiration. Cloud providers publish specifications of their services, which are different style and which makes it hard to use them as a common joint interface. We looked at several existing multi-cloud libraries, including Apache jclouds, DeltaCloud, Kloudless, SecureBlackBox, Temboo and SimpleCloud.

A key requirement was flexibility, which would allow our library to be adapted to changing services or completely different services. We decided to construct our

broker based a combination of proven design patterns, adapted to the context here. Patterns and principles from different libraries were adopted:

- In this vein, we decided to adopt an approach that was also followed in the Apache jclouds library to provide abstraction. Apache jclouds provides cloud-agnostic abstraction [27]. The principle is to use a single instance context for the mapping of a user request.
- The concept of a class for each provider across different levels of services was adopted from a similar design that we found in the SecureBlackBox library.
- The structural pattern building around a manager interface layer at each component level was adopted from the Apache LibCloud architecture.

4 The Ontological Framework for Cloud Storage

We discuss the guiding problems and principles, before introducing our storage abstraction ontology that organises the API architecture and showing how provider functionality is mapped onto this.

4.1 Abstraction, Interoperability and Extensibility

The architecture of our API is built on multiple layers of abstraction. Abstraction serves here to reduce complexity. It provides for service-neutral functional logic which also realises extensibility, i.e., allows additional vendors to be supported without changing the underlying core functional logic of the API design. In the future, new storage services can be added to the API without code change [28]. A programmable abstraction layer provides flexibility to connect and configure services [29]. Thus, interoperability and portability can be achieved. Such APIs are used for developing cloud-based applications like content delivery platforms and back-up applications, as our earlier use case demonstrates.

The main objective of the cloud storage abstraction API is to produce an effective multi-cloud delivery model, with a single portable view that supports enhanced business capabilities such as brokerage [36].

The advantage of bringing these functionalities to an interoperable multi-cloud application provides (i) an easy way of importing and exporting data, (ii) choice over price, (iii) enhanced SLA, and (iv) the elimination of vendor lock-in. While there are standardisation frameworks in this context such as the Cloud Infrastructure Management Interface (CIMI) and the Open Cloud Computing Interface (OCCI) that target interoperability, our integration broker provides interoperability based on an extensible API.

4.2 Storage Abstraction Ontology

An ontology-based layered architecture serves to provide interoperability and extensibility. At the core of the architecture is a storage abstraction ontology that describes the common service concepts across the abstraction layers. This ontology model consists of four main layers, namely Service, Provider, (Level-2) Composite Object and (Level-1) Core Object.

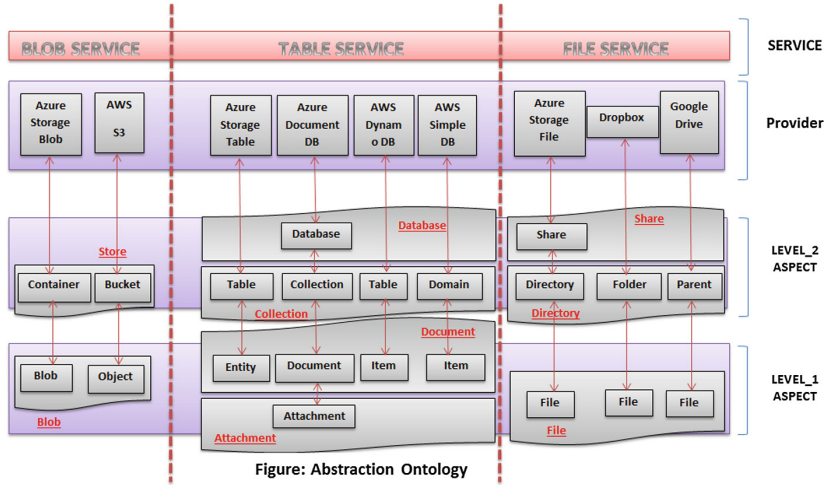


Fig. 2. Storage abstraction ontology based on 4 layers.

- Service: The Service layer is the top layer and is directly integrated into the user interface layer. This layer basically describes the services types supported by the abstraction API, which are blob, table and file service.
- Provider: The Provider layer is the next layer where context object parameters are mapped to the service layer. We supports four main providers – Microsoft Azure, Amazon Web Services, GoogleDrive and DropBox:

| Service | Provider |
|---------|---|
| Blob | Azure storage blob; AWS S3 |
| Table | Azure storage Table; Azure DocumentDB; AWS DynamoDB; AWS SimpleDB |
| File | Azure storage file; DropBox and GoogleDrive |

- Composite Objects – Level-2: The Composite Objects (object level-2) represents the first level (or higher level) of abstraction for the object types blob, table and file. It is service-neutral and established a common naming across individual providers and their specific functionalities. Each layer is abstracted based on common operations and of how the main function is applied in that particular service. Common naming allows to easily categorise storage resources and group them to simplify development.

The Abstraction Ontology diagram (Fig. 2) shows the concept. The blob service has a “Store” concept, which groups for instance ‘Container’ from Azure Storage Blob and ‘Bucket’ from AWS S3. The table service has two different sub-layers – where “Database” belongs to Azure DocumentDB Database, and where “Collection” groups ‘Table’ from Azure Storage Table, ‘collection’ from Azure DocumentDB Collection, ‘Table’ from AWS DynamoDB and ‘Domain’ from ‘AWS SimpleDB’. The file service has two sub-layers where “Share” belongs to Azure Storage File and “Directory” groups ‘Directory’ from Azure Storage, ‘Folder’ from DropBox and ‘Parent’ from GoogleDrive.

- Core Objects – Level-1: The Core Object (object level-1) aspect represents the lower level of storage object abstraction. This layer contains the core functionalities of a particular service across different providers.

The classes at this level are extended from an abstract class called AbstractConnector, implementing the abstractor pattern. The class implements the abstract methods defined in an AbstractConnector class. The mapping from Level-2 to Level-1 is performed by an interface class called Manager. This Manager identifies the provider class by its key. Basic CRUD operations on the storage resources are core methods. In order to implement these functions, each operation “request” should “pass through” the Level-2 mappings and is then mapped across the service and providers.

The blob service has “Blob”, which groups ‘Blob’ from Azure Storage Blob and ‘Object’ from AWS S3. The table service has two sublayers. It has an “Item” to group ‘Entity’ from Azure Storage Table, ‘Document’ from Azure, ‘Item’ from AWS DynamoDB and ‘Item’ from AWS SimpleDB. Furthermore, the second sublayer “Attachment” belongs to the Azure DocumentDB Attachment. The file service has “File” grouping ‘File’ from Azure Storage File, ‘File’ from DropBox and ‘File’ from GoogleDrive.

4.3 Storage Service Provider Functionality

An important concern was having common naming for mapping the user’s requests onto the different services. The broker acts as an adapter for accessing different providers’ services through a common interface. We found a high degree of commonality between different cloud provider functions and their names. However, some operations exist in one provider, but not in others. Furthermore, the parameters in some of the methods also differ between providers.

In the example below, the Level-2 aspect “Store” in the Blob service supports providers with different specific names, namely Azure’s storage blob container and AWS’s S3 bucket. The table also shows the common createStore() method and the corresponding Azure-specific and AWS-specific underlying method calls. The approach is based on identifying synonyms for common object names, such as container (Azure) and bucket (S3) for ‘store’:

| Common name (Level-2) | Name in azure storage blob | Name in AWS S3 |
|-----------------------|----------------------------|----------------|
| Store | Container | Bucket |

The same then applies to function names:

| Common method name | Name in azure storage blob | Name in AWS S3 |
|--------------------|----------------------------|------------------------------|
| createStore() | create() create container | createBucket() create bucket |

The abstraction ontology maps similar service groupings together across different cloud providers. Selected services have similar or the same core functional logic – grouped into levels in the ontology. Based on this, the framework design includes the “service”, “provider”, “composite object” and “core object” for its implementation.

In the example below, the common level-2 composite “Collection” is already defined for two providers. There are four corresponding service names: Azure

storage table ‘table’, AWS document DB ‘collection’, AWS dynamo DB ‘table’ and AWS simple DB ‘domain’. The table below shows the common `getCollectionMetadata()` method and its corresponding provider API method calls:

| Common ontology name | Azure storage table | Azure Document DB | AWS Dynamo DB | AWS Simple DB |
|---|---|---|--|--|
| Composite: Collection | Table | Collection | Table | Domain |
| Operation: <code>getCollectionMetadata()</code> | - | - | <code>describeTable()</code> returns information about the table. | <code>domainMetadata()</code> returns information about the domain. |
| Operation: <code>listCollection()</code> | <code>listTables()</code> Lists the table names in the account | <code>readCollection()</code> Reads a document collection by the collection link | <code>listTables()</code> Simplified method for invoking <code>ListTables</code> | <code>listDomains()</code> Lists all domains associated with the Access Key ID |

The method name to retrieve the metadata of a collection in the Table service is not supported by Azure, but AWS does. So, a common operation can not be realised across the level-2 composite Collection. A similar problem exists with the blob and file services. This lack of consistency in the available provider API operations has led to the omission of valuable API method calls. We also introduce another example, which is the common `listCollection()` method and its corresponding provider API method calls. The method name to retrieve the list of collections in the table service is supported by both Azure and AWS. However, the method names are different, although description and logic are the same.

5 Design of the Cloud Storage API

Our design involves a mapping of an ontology-based conceptual framework onto a layered architecture, which in turn was structured by suitable design patterns. We look at security management of the different service providers.

5.1 An Ontology-Driven Architecture

We applied a “model-driven” software engineering technique to simplify the process of design from concept modeling to implementation. This was done at each level in the abstraction ontology by breaking entities into single components. Adopting this kind of best-practice in software design was important in order to reduce the development overhead and produce a quality library that can be extended easily. Two types of modeling approaches could have been used here.

- Firstly, a provider-specific model, in which the provisioning and deployment of the abstraction library is defined for each cloud provider.
- Secondly, a cloud provider-independent model, which defines the provisioning and deployment of the abstraction library in a cloud-agnostic way.

We adopted the approach taken in the CloudML EU-funded research project. There, a domain-specific modelling language is used to reduce the complexity of cloud system design. CloudML enables to provision and deploy an abstraction library. Its design includes what we call level-1 core objects, which are assembled

based on the CloudML internal component design. These are mapped to level-2 components by using a model-driven approach. A client using the service does not necessarily know about the internal deployment, and there is no limitation on the design and evolution of the multi-cloud abstraction library.

5.2 Application of Design Pattern

Design patterns play a central role in organising the layered ontology-based architecture in order to achieve the required maintainability and extensibility, but also in general the quality of the software.

Mapping Based on an Object Context for Maintainability. For any multi-cloud library design patterns can reduce the need to have an object instantiation for each provider's class using the constructor. This was a problem noted for the jclouds library. A lack of code clarity and high level of complexity in the framework pattern was observed.

In our API, in order to avoid this problem and to provide a stable, maintainable code base, the context builder class is added to the architecture. This builder class includes a key and a value parameter pair. This pair is called an item, which adds the service, the provider, the aspect key, the operation key and the input parameters to the context. Then, this context object is passed on to execute the API method call. This mapping is applied for all the services supported by our API. Below, we outline the mapping of parameters into a single context instance.

```
Context context = new Context();
context = addServiceContext(context);
context = addServiceProviderContext(context);
context = BlobService.addParameters(
    IConstants.ASPECT_KEY, IConstants.LEVEL-2_STORE, context);
context.addItem(new Item(
    IConstants.OPERATION_KEY, IConstants.OPERATION_CREATE));
context = BlobService.addParameters(
    IConstants.STORE_NAME, storeName, context);
```

Extensibility Through a Plug-in Framework. API design principles state that a developer should not have visibility of the underlying low-level abstraction classes, interfaces and methods. If a future extension can support new features and services, then the framework should not have to be redesigned or its behavior changed. We say that the framework acts like a plug-in for any new features, services or providers. We achieve this in the design by enforcing that an abstract class cannot be instantiated, it can only be inherited, as a strict coding rule.

The level-1 layer, which implements the lower-level API methods, is extended from the AbstractConnector class. This abstract class must implement the interface IConnector and all of its associated methods. The reason for this is because an abstract class, by definition, is required to create subclasses of its instance. The subclasses are required by the compiler to implement any interface methods

that the abstract class has left unimplemented. Less effort is required to extend the API because the framework itself remains unchanged in that case.

Multi-service Support Through a Manager Interface Layer. The level-2 layer is limited because of the separation of the user level request which is meant to distinguish between different API methods provided by the same provider. For example, AWS provides DynamoDB and SimpleDB. Similarly, Azure provides storage table and DocumentDB services. In order to remedy this, an interface component called manager is implemented. The manager is responsible for identifying the corresponding “aspect-key” that is encapsulated within the context parameter discussed earlier on.

We use two types of managers: the store manager and the table manager.

- The composite object Level-2 aspect is the higher level of abstraction. Since Level-2 helps to identify the differences between the services, the store manager interface is added in this layer, which splits the request to either blob or file or table service at core object level-1.
- The table manager is used in a similar way. For example, the table service at Level-1 has two APIs, the DynamoDB and SimpleDB, supported by one provider, AWS. In order to differentiate between the services, an interface component called manager was added to identify the common method name.

The relationship between the abstract class and the core logic of the level-2 aspect is managed using the context parameter.

5.3 Apache jclouds and Design Patterns

Our multi-cloud storage abstraction layer was designed using some of the design concepts and patterns of jclouds. Apache jclouds is an open source library available in Java and Clojure, which supports several major cloud providers. The jclouds library offers both a portable abstraction framework as well as cloud-specific features. The main aim of jclouds is to manage errors, concurrency and cloud complexity better.

The jclouds Architecture. jclouds features of a portable abstraction layer called ‘View’, responsible for splitting the service type and cloud provider. A ‘View’ is connected to a provider-specific API or library driven API. The Context Builder class maps the context object along with its parameters. The parameters include provider class object, view, API metadata and provider metadata. This object will be bound as a *singleton object* called Context and it is passed to the context builder. The API Metadata class populates friendly names for the key, which has two values – the type and the view information. The Service Registry acts like a manager, which is responsible for holding the key to connect to a provider’s class. The framework implements a *builder pattern* for request and response, which connects to a backend API, along with authentication.

In the context of our broker, the jclouds library caters for blob and compute services. The following code block outlines the jclouds library code for calling a context for an Azure blob. It uses the context builder class. The basic concept of abstraction used in the jclouds library is based on the builder design pattern known from software engineering. A context with service provider Azure that offers the portable BlobStore API would look like as follows:

```
BlobStoreContext context =
    ContextBuilder.newBuilder("azureblob")
        .credentials(storageAccountName, storageAccountKey)
        .buildView(BlobStoreContext.class);
```

5.4 Security Analysis – Authentication Mechanism

Security is another concern that needs to be unified across the providers in addition to the mapping of concepts for core and composite storage objects used by the different providers into a common ontology. Authentication, however, differs across the providers selected.

1. The authentication in *GoogleDrive* is based on a `client_secret` json file. A project is created in the Google developer's console. The Drive API and OAuth protocol is enabled. The credentials are generated and saved as a `client_secret` json file. In the coding, the authentication method should have the permission scope and drive scope set to 'GRANT'. When the browser opens for the authentication response, the client is permitted full read and write access.
2. The authentication in *DropBox* uses an access token, which is generated in the console. An application is created under the app console and its permission is set to 'FULL'. Later, the authentication is set by linking the account using the access token when passing the instance of the API client.
3. The authentication in *Microsoft Azure* is based on an account subscription that allows for the accessing of the resources available within an azure account. The Blob service is provided within an Azure storage account. The azure storage account name, also known as namespace, is the first level for processing authentication to the services within the storage account (blob, file and table). It uses token-based authentication. The authentication of the Azure storage blob is done using a connection string which has the parameters of the storage account name and primary key. Similarly, the Table service is supported within an Azure storage account. The authentication of the Azure storage table is done using a connection string which has the parameters of the storage account name and primary key. An Azure Document DB account is required for accessing the Azure Document Db service and requires a master key and URI (end-point).
4. The authentication in *Amazon Web Services* is based on a secret key and an access key, which is common across all services supported by our storage API. An AWS user should have a specified role with the required resource access permissions. Identity Access Management allows the user to set the role and access privileges, and this provides each user with sufficient credentials.

The account can be activated using phone verification and authentication. Credential auditing and usage reports can be used for review purposes.

The different authentication processes were considered for the broker authentication method, i.e., calls from the multi-cloud storage abstraction API. Our solution is based on a credentials object:

- Credentials storage: The credentials are stored in a common config file. So, the user is not shown the authentication part as it happens in the back-end.
- Credentials update: If the credentials need to be changed, they are only changed in the configuration file, which reduces overhead for the user to set up the authentication process.

6 Discussion and Conclusions

We have discussed maintainability and extensibility as central objectives that need to be evaluated here throughout the architecture discussion in the previous section. In a summarising discussion, we return here to a few important concerns such as establishing testability and maintainability through suitable design patterns, e.g., the dependency injection pattern, to point out benefits. Other aspects have already been discussed throughout Sect. 5 above.

Design Patterns and Software Quality. From the discussion above, specifically the code, it can be clearly seen that jclouds gets a separate instance for each provider’s class and, in some cases, it makes direct REST calls to the underlying provider API. Thus, the programming style in jclouds follows the *dependency injection software design pattern*. It uses two *programming frameworks*: firstly, Google Guice, which is a Google library alternative to Spring, and, secondly, Guava, which supports transformation, concatenation and aggregation for storage services.

Another quality concern shall be addressed: recompilation overhead. Dependency injection avoids code duplication, is unit-testable and modular. It thus allows injecting of the service class instead of calling the API service method, achieved by writing custom code and connecting it at run time, which avoids recompiling. Custom code instantiates an object for each service and provider.

Quality Factors – Testability and Extensibility. Testing is a further concern. The jclouds library uses dependency injection, as discussed above, which makes reference to an object before it will proceed for execution. Implementing dependencies by constructors, using the ‘new’ constructor may result in difficulties for unit testing. Performing dependency injection using a *factory method* is a traditional solution to the testing concern.

This is also known as indirect dependency, where the factory method is realised by having an interaction class between the client class and the service class. It was considered that the use of too many interaction classes would make

the code more complex and result in tight coupling between the abstraction layers. This would hide the definition of abstraction and furthermore, it would not facilitate the future extension of the library.

According to the principles of API design, there should be a small number of functionalities shared across the entire cloud provider API. This has been achieved in the abstraction design used.

Final Comments. The aim of cloud service brokerage is customising or integrating existing services or making them interoperable. Following the classification schemes in [12, 13], we have developed an integration broker:

- the main purpose is intermediation between cloud consumers and providers to provide advanced capabilities (interoperability and portability),
- it builds up on an intermediary/broker platform to provide a marketplace to bring providers and customers together,
- the broker system type is a multi-cloud API library.

We presented here on a broker solution [1] for cloud storage service providers to implement a joint interface to allow

- portability and migration for the user, i.e., the consumer of the services,
- extensibility for the broker provider to allow changed or new services to be included.

Our broker enables through its joint API also the opportunity for a cloud storage user to easily migrate between services or to use multiple services at the same time, depending on preferred characteristics such as security or performance [5].

Many broker implementations and multi-cloud APIs exist. We provide a novel view by focussing here on the construction of a broker API and looking at software architecture principles behind it. Again, ontologies have been used before, but we demonstrate here how a layered ontology and a corresponding layered architecture together with the use of appropriate design patterns can better help to achieve extensibility and efficiency of the implementation. The selection of design patterns has a significant impact on the testability, maintainability and extensibility of the layered architecture that we have developed.

As future work, we plan to extend the broker by adding further services by other providers to empirically verify the extensibility of the library. While our API-based architecture only supports public cloud providers, this can be extended to include private clouds in future. A more long-term usage beyond some performance testing on the provider services, should also help to better judge the maintainability in addition to the expected positive affect from the pattern application. More work could also go into more uniform specification of cloud services [25, 35] aiming at more standardisation of the interfaces.

Acknowledgements. This work was partly supported by IC4 (Irish Centre for Cloud Computing and Commerce), funded by EI and the IDA.

References

1. Ried, S.: Cloud Broker – A New Business Model Paradigm. Forrester, Cambridge (2011)
2. Benslimane, D., Dustdar, S., Sheth, A.: Services mashups – the new generation of web applications. *Internet Comput.* **12**(5), 13–15 (2008)
3. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., Morrow, M.: Blueprint for the inter-cloud: protocols and formats for cloud computing interoperability. In: *International Conference on Internet and Web Applications and Services* (2009)
4. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) *ICA3PP 2010. LNCS*, vol. 6081, pp. 13–31. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13119-6_2
5. Elango, D.M., Fowley, F., Pahl, C.: Testing and comparing the performance of cloud service providers using a service broker architecture. In: Mann, Z.Á., Stolz, V. (eds.) *ESOCC 2017. CCIS*, vol. 824, pp. 117–129. Springer, Cham (2018)
6. Cloud Standards (2017). <http://cloud-standards.org/>
7. ETSI Cloud Standards (2017). <http://www.etsi.org/newsevents/news/734-2013-12-press-release-report-on-cloudcomputing-standards>
8. Fehling, C., Mietzner, R.: Composite as a service: cloud application structures, provisioning, and management. *Info. Technol.* **53**(4), 188–194 (2011)
9. Pahl, C., Jamshidi, P., Weyns, D.: Cloud architecture continuity: change models and change rules for sustainable cloud software architectures. *J. Softw. Evol. Process* **29**, e1849 (2017). <https://doi.org/10.1002/smr.1849>
10. Pahl, C., Jamshidi, P., Zimmermann, O.: Architectural principles for cloud software. *ACM Trans. Internet Technol.* (2018, to appear)
11. Forrester Research: Cloud Brokers Will Reshape The Cloud (2012). <http://www.cordys.com/ufc/file2/cordyscmssites/download/09b57cd3eb6474f1fda1cfd62ddf094d/pu/>
12. Fowley, F., Pahl, C., Zhang, L.: A comparison framework and review of service brokerage solutions for cloud architectures. In: Lomuscio, A.R., Nepal, S., Patrizi, F., Benatallah, B., Brandić, I. (eds.) *ICSOC 2013. LNCS*, vol. 8377, pp. 137–149. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06859-6_13
13. Fowley, F., Pahl, C., Jamshidi, P., Fang, D., Liu, X.: A classification and comparison framework for cloud service brokerage architectures. *IEEE Trans. Cloud Comput.* (2017). <https://doi.org/10.1109/TCC.2016.2537333>. <http://ieeexplore.ieee.org/document/7423741/>
14. Javed, M., Abgaz, Y.M., Pahl, C.: Ontology change management and identification of change patterns. *J. Data Semant.* **2**(2–3), 119–143 (2013)
15. Garcia-Gomez, S., et al.: Challenges for the comprehensive management of cloud services in a PaaS framework. *Scalable Comput. Pract. Exp.* **13**(3), 201–214 (2012)
16. Elango, D.M., Fowley, F., Pahl, C.: Using a cloud broker API to evaluate cloud service provider performance. Research report 471, Department of Informatics, University of Oslo, pp. 63–74 (2017)
17. Gartner: Cloud Services Brokerage. Gartner Research (2013). <http://www.gartner.com/it-glossary/cloud-servicesbrokerage-csb>
18. Grozev, N., Buyya, R.: InterCloud architectures and application brokering: taxonomy and survey. *Softw. Pract. Exp.* **44**, 369–390 (2012)
19. Pahl, C., Jamshidi, P.: Microservices: a systematic mapping study. In: *Proceedings CLOSER Conference*, pp. 137–146 (2016)

20. Taibi, D., Lenarduzzi, V., Pahl, C.: Processes, motivations and issues for migrating to microservices architectures: an empirical investigation. *IEEE Cloud Comput.* **4**(5), 22–32 (2018). <http://ieeexplore.ieee.org/document/8125558/>
21. Hofer, C.N., Karagiannis, G.: Cloud computing services: taxonomy and comparison. *J. Internet Serv. Appl.* **2**(2), 81–94 (2011)
22. Jamshidi, P., Sharifloo, A., Pahl, C., Arabnejad, H., Metzger, A., Estrada, G.: Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In: *12th International ACM SIGSOFT Conference on Quality of Software Architectures QoSA* (2016)
23. Arabnejad, H., Jamshidi, P., Estrada, G., El Ioini, N., Pahl, C.: An auto-scaling cloud controller using fuzzy Q-learning - implementation in openstack. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) *ESOCC 2016*. LNCS, vol. 9846, pp. 152–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_10
24. Gacitua-Decar, V., Pahl, C.: Structural process pattern matching based on graph morphism detection. *Int. J. Softw. Eng. Knowl. Eng.* **27**(2), 153–189 (2017)
25. *IEEE Cloud Standards* (2015). <http://cloudcomputing.ieee.org/standards>
26. Jamshidi, P., Ahmad, A., Pahl, C.: Cloud migration research: a systematic review. *IEEE Trans. Cloud Comput.* **1**, 142–157 (2013)
27. jclouds: jclouds Java and Clojure Cloud API (2015). <http://www.jclouds.org/>
28. Ferrer, A.J., et al.: OPTIMIS: a holistic approach to cloud service provisioning. *Future Gener. Comput. Syst.* **28**(1), 66–77 (2012)
29. Konstantinou, A.V., Eilam, T., Kalantar, M., Totok, A.A., Arnold, W., Snibler, E.: An architecture for virtual solution composition and deployment in infrastructure clouds. In: *International Workshop on Virtualization Technologies in Distributed Computing* (2009)
30. Pahl, C.: Layered ontological modelling for web service-oriented model-driven architecture. In: Hartman, A., Kreische, D. (eds.) *ECMDA-FA 2005*. LNCS, vol. 3748, pp. 88–102. Springer, Heidelberg (2005). https://doi.org/10.1007/11581741_8
31. Pahl, C., Giesecke, S., Hasselbring, W.: Ontology-based modelling of architectural styles. *Inf. Softw. Technol.* **51**(12), 1739–1749 (2009)
32. Mietzner, R., Leymann, F., Papazoglou, M.: Defining composite configurable SaaS application packages using SCA Variability Descriptors and Multi-tenancy Patterns. In: *International Conference on Internet and Web Applications and Services* (2008)
33. Pahl, C., Xiong, H.: Migration to PaaS clouds - migration process and architectural concerns. In: *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA* (2013)
34. Pahl, C., Xiong, H., Walshe, R.: A comparison of on-premise to cloud migration approaches. In: Lau, K.-K., Lamersdorf, W., Pimentel, E. (eds.) *ESOCC 2013*. LNCS, vol. 8135, pp. 212–226. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40651-5_18
35. Papazoglou, M.P., van den Heuvel, W.J.: Blueprinting the cloud. *IEEE Internet Comput.* **15**, 74–79 (2011)
36. Petcu, D., et al.: Portable cloud applications—from theory to practice. *Future Gener. Comput. Syst.* **29**(6), 1417–1430 (2013)
37. Amazon Simple Storage Service (S3) Cloud Storage AWS <https://aws.amazon.com/s3/>
38. Dropbox. <https://www.dropbox.com/>
39. Azure Storage - Secure cloud storage. <https://azure.microsoft.com/en-us/services/storage/>

40. Google Drive - Cloud Storage & File Backup. <https://www.google.com/drive/>
41. Jamshidi, P., Pahl, C., Mendonca, N.C.: Pattern-based multi-cloud architecture migration. *Softw. Pract. Exp.* **47**(9), 1159–1184 (2017)
42. Pahl, C., Brogi, A., Soldani, J., Jamshidi, P.: Cloud container technologies: a state-of-the-art review. *IEEE Trans. Cloud Comput.* (2017). <https://doi.org/10.1109/TCC.2017.2702586>. <http://ieeexplore.ieee.org/document/7922500/>
43. Aderaldo, C.M., Mendonca, N.C., Pahl, C., Jamshidi, P.: Benchmark requirements for microservices architecture research. In: 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering. IEEE (2017)
44. Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L.E., Pahl, C., Schulte, S., Wettinger, J.: Performance engineering for microservices: research challenges and directions. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (2017)



Cloud-Native Databases: An Application Perspective

Josef Spillner^(✉) , Giovanni Toffetti, and Manuel Ramírez López

Service Prototyping Lab, School of Engineering,
Zurich University of Applied Sciences, 8401 Winterthur, Switzerland
{josef.spillner,toff,ramz}@zhaw.ch
<http://blog.zhaw.ch/icclab/>

Abstract. As cloud computing technologies evolve to better support hosted software applications, software development businesses are faced with a multitude of options to migrate to the cloud. A key concern is the management of data. Research on *cloud-native applications* has guided the construction of highly elastically scalable and resilient stateless applications, while there is no corresponding concept for *cloud-native databases* yet. In particular, it is not clear what the trade-offs between using self-managed database services as part of the application and provider-managed database services are. We contribute an overview about the available options, a testbed to compare the options in a systematic way, and an analysis of selected benchmark results produced during the cloud migration of a commercial document management application.

1 State Management in Cloud-Native Applications

Cloud-native applications (CNA) are software applications which pass down beneficial cloud computing characteristics. They use cloud platform and infrastructure services to become executable, offer their own functionality as software service interfaces, are resilient against dependency service unavailability and other incidents, scale elastically with user requests, are always available on demand and are billed with a pay-per-use utility scheme without upfront cost [2]. The inherent service orientation required for CNA favours a microservices model with explicitly stateful and stateless services. The handling of data is confined to the stateful services. These must in turn be highly available and resilient to prevent loss, corruption or delay of data operations. Databases, message queues, key-value stores, filesystems and other data access models have been analysed in prior works concerning these requirements [7, 13, 14]. The desired characteristics depend on near-instant service replication [10] which implies consistent data replication and sharding mechanisms.

Figure 1 shows a typical topology of stateful and stateless microservices orchestrated to offer a single application as a service in a highly available and resilient manner on top of plain cloud infrastructure services. Almost all approaches rely on coordinated replication which brings self-awareness about

its role (e.g., master or slave) to each microservice. Furthermore, they rely on fast-spawning service implementations (e.g., containers or light-weight hypervisors) to achieve rapid elasticity upon request spikes and instance recovery after crashes.

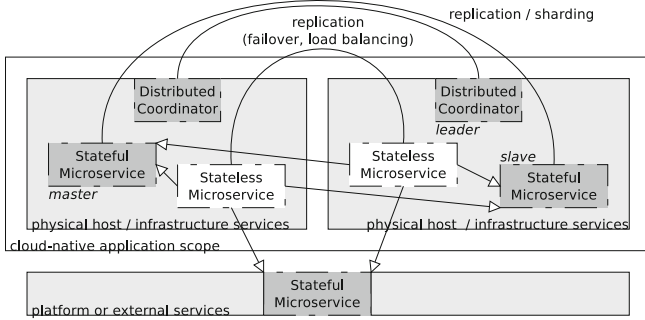


Fig. 1. Cloud-native application with internal and external data management

It is however not clear under which circumstances applications should manage data by themselves. The range of commercially offered platform-integrated stateful services is increasing. Their common value proposition can be expressed in a simplified way by *paying more to manage less*. But for a business decision, the value needs to be quantified. Due to the multitude of possible options, businesses need to obtain metrics on which such decisions can be performed. Apart from the pricing, such decisions need to account for risks and for end-to-end service provisioning quality and effort which from a software engineering perspective always includes the consideration of client-side bindings to the services.

To reduce the problem scope, we limit the research to applications which handle large structured documents in database systems. Hence, through this paper, empirical studies of how databases operated in a cloud-native context behave in commercial cloud environments are made possible. The main contribution is a testbed to measure and compare different database options from a vendor-neutral perspective. The resulting distinction between self-managed and provider-managed databases covered by the testbed is expressed in Fig. 2.

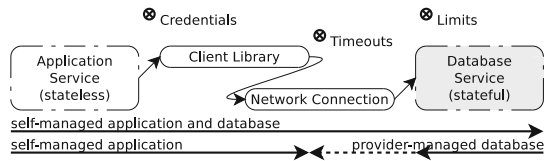


Fig. 2. Scopes of cloud-native database management

The paper is structured as follows: First, it presents the possible options for cloud-native databases, differentiating between fully managed and application-controlled offers. Then, it defines a method to compare and rate cloud database services both on the technical and on the pricing level. The method translates into a design of a testbed whose architecture and implementation we present. Experiments we have conducted based on this method are then explained together with the obtained results. The findings from the experiments need to be interpreted in alignment with business strategies. For this reason the paper concludes with an open discussion about the strategic impact of using the testbed in a systematic way during the application engineering process.

2 Cloud-Native Database Options

Apart from conventional relational or document-centric databases, the migration of applications into the cloud and the associated new operational requirements have led to novel design choices and along with them new research challenges. Recent database models thus include encrypted, privacy-preserving and stealth databases, energy-efficient database operators and adaptive query systems over dynamically provisioned resources [4, 8]. Few of these designs have progressed beyond prototypical systems, but from an applied science perspective, we are interested in what recommendations can be given to application developers today. Hence, only conventional (relational and document-centric) database systems and services which are widely available on cloud platforms are considered along with systems commonly described as cloud-enabled or cloud-ready.

We distinguish between the choices of database hosting primarily by the responsibility. Database management systems can be managed by the application provider as part of the application (e.g. as application-controlled container), outside of the application scope itself (e.g. as a separate virtual machine whose autoscaling is determined by cloud facilities), and as fully cloud-managed service, typically named Database-as-a-Service (DBaaS). Our focus is on *cloud-native database* (CNDB) options which adhere to expectations from cloud application developers such as elastic scaling, resilience against unexpected issues, flexible multi-tenancy isolation and high performance at low price.

2.1 Self-managed Database Systems and Microservices

The widespread proliferation of open source database management systems has led to the inclusion of these systems into software applications. The application logic then controls the lifecycle of the database, launches and terminates it as needed, and directly accesses it, often without authentication or through a single user account. Tenants in the application are in this case mapped to the database through identifiers or unprivileged separation such as tables or columns.

Cloud-native applications are often decomposed into horizontally scalable microservices where all instances are of equal importance in a peer structure. Only few database systems are currently mirroring this ability. Many still require

a master-slave setup where the master instance needs to be launched before the slave instances and must never fail, or variants thereof with multi-master replication. We analyse selected database systems concerning their use as disposable microservices in Table 1. Of these, only Crate fully conforms to this model, although a technology preview also exists for MongoDB (for master-slave replication).

Table 1. Available self-managed database microservices

| Name | Relation of instances |
|------------|---|
| CouchDB | Master-slave and master-master replication, manual sharding |
| MongoDB | Replica sets with master-slave replication, keyed sharding |
| Crate | Set of peers with automated sharding upon scaling |
| PostgreSQL | Master-slave replication, sharding through Citus |
| MySQL | Master-slave replication, sharding through Fabric |

2.2 Provider-Managed Database Services

From a cloud application perspective, it is desirable to maximise the flexibility by freely choosing among application-controllable software and managed services for the assumed database interface. Despite efforts to standardise the interfaces for database-as-a-service (DBaaS), the implementation differences are significant enough to warrant the propagation of information about the underlying database system. For instance, a developer may know how to write SQL statements but can optimise them and avoid pitfalls when knowing that the engine behind the SQL interface is in fact a MariaDB 10.2 with the XtraDB storage engine. This knowledge should be conveyed and flexibly interpreted using discoverable service descriptions, but in practice, it is often tightly coupled to the application. Furthermore, database interface and implementation options provided in the commercial cloud space vary significantly. Table 2 compares the availability of database interfaces at six public cloud (platform) providers from two countries, USA and Switzerland. Implementations marked with asterisk are available as open source and thus allocatable for local testing by application developers prior to paying for the cloud deployment.

Despite multi-database service offers by most providers, the table is sparse. This means that vendor lock-in risks need to be assessed. Furthermore, the pricing of DBaaS differs for offers with the same interface. For instance, MongoDB services are offered by Microsoft (as interface adapter to CosmosDB) and by the Swisscom Application Cloud (AC). The Swisscom offer excluding high availability starts at CHF 12 per month including 1 GB storage and 256 MB RAM. The equivalent Azure offer (hosted in Europe-West) starts at CHF 134 but includes 10 GB storage and 5 DTUs, a custom unit expressing the processing power. For an application engineer who wants to process a data volume of 1 GB, it is

Table 2. Available provider-managed database services

| Amazon Web Services | Google Cloud | Microsoft Azure | IBM Bluemix | APPUiO | Swisscom AC | Application Interface | Implementation |
|---------------------|----------------|------------------|----------------|--------|-------------|---------------------------------|----------------|
| X ^a | X ^b | | | | | SQL | MySQL * |
| | | | | X | X | | MariaDB * |
| | | | X | X | | | PostgreSQL * |
| | | | | | | | Aurora |
| | | | | | | | Oracle DB |
| | X ^c | | | | | | SQL Server |
| | | X | | | | | DB2 |
| | | (X) ^d | | X | X | JSON QL or similar (Mango etc.) | MongoDB * |
| | | | X ^e | | | | CouchDB * |
| X | | | | | | | DynamoDB |
| | | X | | | | | CosmosDB |
| | | X | | | | Other | TableStorage |
| | X | | | | | | Datastore |
| | X | | | | | | BigTable (*) |
| X ^f | X ^g | X | | | X | | Redis * |

Notes: ^aRDS, ^bCloud SQL, ^cDatabase Service, ^dCosmosDB adapter, ^eAs Cloudbant NoSQL, ^fAs ElastiCache, ^gVia external RedisLabs service

not clear if the cheaper offer would be performance-wise on par without further experiments. There are detailed studies on cloud database services in general [11]. In contrast, our focus is on their suitability for cloud-native applications.

3 Comparison Method and Testbed

The automatable comparison of databases is rooted in two main characteristics: performance and resilience. Other metrics such as price and isolation can be derived from trace data in conjunction with external information. Several queries and transactions are run to measure the performance through an application-specific benchmark. It includes the preparation of structures (tables, collections), individual inserts, bulk inserts, queries and deletions. Furthermore, the availability is measured and in the case of self-managed database services actively impeded by controlled interference and termination, leading to data about the resilience.

As our chosen approach is to provide a testbed to compare database options, its functional and non-functional requirements need to be defined first. The functional requirements are:

1. The testbed must run itself in the target environment of the cloud-native application to yield realistic metrics with simple queries and complex transactions.
2. Both self-managed and provider-managed database services need to be supported.

3. The testbed operator must be able to choose the dataset under test, either an existing one or a synthetic one which is generated as part of the operation.

The non-functional requirements are:

1. The scale of testing needs to be configurable to balance representative and timely results. Therefore, the runtime needs to be chosen to range from mere minutes to multi-day sampling.
2. All tests need to be idempotent to allow for repetitions and statistical detection of anomalies.

3.1 Testbed Architecture and Implementation

The testbed architecture is derived from the requirements. To correlate with cloud-native applications, a containerised approach is taken. Both the testbed itself, with its performance benchmark and resilience calculation parts, and all self-managed database services are launched as container compositions. Figure 3 visualises the technique of how the experiments are conducted by using Docker Compose as orchestrator of containers. One container contains a performance benchmark application, another one a fault provocation application, two stateful containers serve as persistent input and output volumes for the reference dataset and the results respectively, and additional containers spawn the database systems. The testbed containers allow for parameterisation through environment variables to override any values in the internal configuration file. The most important properties include binding metadata and credentials. Furthermore, the testbed supports five configurable multi-tenancy isolation levels.

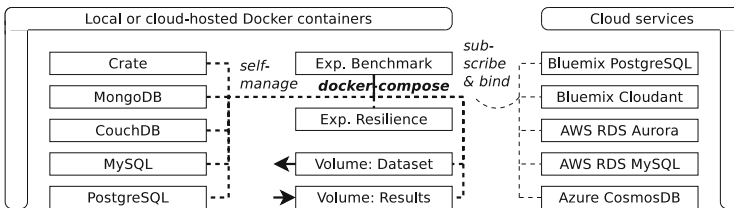


Fig. 3. Orchestrated containers and services as part of the experiment setup

Our implementation of this architecture is called CNDBbench, focusing on the benchmarking part while also containing the resilience part. It is consisting of Python classes for all supported database interfaces and the Docker image generation scripts, and is made available as open source software for use in other migration cases (see Repeatability).

3.2 Testbed Preparation: Document Management Scenario

Each instance of the testbed needs to be prepared according to application-specific needs. The guiding objective of our research has been to analyse database options for the class of cloud-native document management applications. Their requirement is storing millions of documents (e.g. scanned PDFs of dozens of MB in size) along with document metadata such as ownership, permissions, audit trails and searchable full text determined by OCR prior to insertion. From the application perspective, the design then involves stateful (database) components which are realised as bindings to database services or instances of application-controlled database microservices. Figure 4 demonstrates a document management scenario and the possible realisation options.

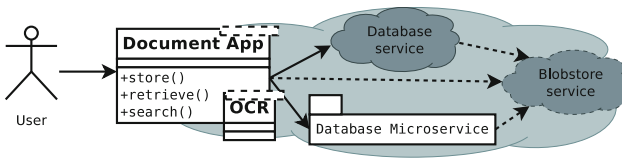


Fig. 4. Document management scenario

The reference dataset to evaluate the database choices consists of 100,000 generated entries which correspond to an actual domain-specific dataset with scanned newspaper articles. With associated metadata such as origin and access control lists, there are 1.4 million entries in total. The medium-sized data with large blob documents and structured metadata is representative for the domain of document management in the cloud through databases; alternative hybrid designs using blob storage are not considered in the present scenario. The following operations are performed to get both performance and deviation metrics: insertion of data, search and retrieval of partial data. This selection matches transactions in typical document management applications where updates and deletions happen rather sporadically.

3.3 Testbed Operation

Once the testbed is prepared, it needs to be operated in a way which most closely corresponds to the eventual operation of the application. Specifically, network delays and latencies as well as microservice execution technologies need to be properly reflected. Figure 5 shows seven testbed configurations which correspond to all possible combinations of how to manage application data in the cloud. More variability is added by defining for the cases of application-managed databases where to physically store the data. Our research assumes attaching volume containers whereas provider-managed storage areas would be another option.

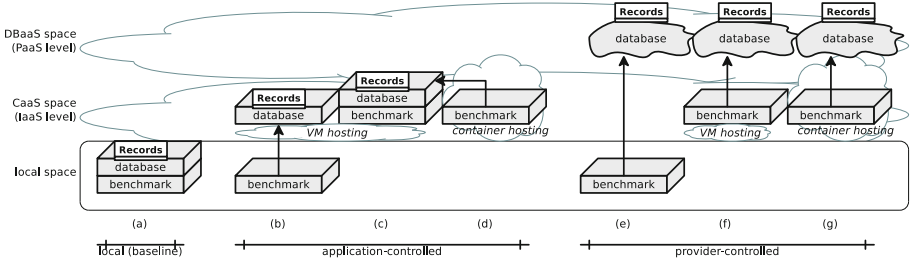


Fig. 5. Combinations of local, application-managed and provider-managed containers with application-managed and provider-managed databases

4 Selected Results

This section reports on results we have obtained from running the testbed in some of the explained operational combinations using the document management dataset. The research on the figurative *cloud-nativeness* of databases have been conducted with experiments targeting the desired technical properties of the specific application domain of document management. In total, 28 experiments have been performed and recorded, showing the versatility of CNDBbench. Selected results concerning performance, multi-tenancy flexibility and pricing will be reported. Apart from the results described here due to interesting observations, all experiments and results are analysed and described in a technical appendix to this paper (see Repeatability).

Five relational and document database systems from Table 2 have been selected for the study of the first group. They are briefly summarised in Table 3. Among those, PostgreSQL and MySQL are relational database systems (albeit with recently added JSON document processing capabilities) and have been

Table 3. Evaluated database system software and cloud services

| Software/service | Data model | Runtime | Distribution |
|--------------------|-------------|------------|-------------------|
| CouchDB | Document | Erlang | Create-sharding |
| MongoDB | Document | C++ | Config-sharding |
| Crate | Mixed-model | Java | Auto-sharding |
| PostgreSQL | Relational | C | Master-sharding |
| MySQL | Relational | C, C++ | Fabric-sharding |
| AWS RDS Aurora | Relational | MySQL | Read-replicas |
| AWS RDS MySQL | Relational | MySQL | Read-replicas |
| Azure CosmosDB | Document | DocumentDB | Key-sharding |
| Bluemix PostgreSQL | Relational | PostgreSQL | Failover-replicas |
| Bluemix Cloudant | Document | CouchDB | None |

available in early versions since the mid-1990s. CouchDB and MongoDB are often-cited representatives for document-centric systems which appeared in the late 2000s. Crate is the most recent system, created in 2014, whose focus on cloud deployments is stressed by masterless distributed operation and automatic node recovery in combination with a standard SQL-over-HTTP interface. It offers a mixed document/column store. All five systems have subtle differences in how they shard (and replicate) data.

For the second group, summarised in the bottom half of the table, three database service providers have been chosen: Amazon Web Service’s Relational Database Service (RDS) with the Aurora implementation, which is a custom storage engine, in addition to the stock MySQL with its InnoDB, MyISAM and other default engines, IBM’s Cloudant NoSQL and PostgreSQL service on its Bluemix platforms, which as the name suggests are a document store and a relational database, respectively, and Azure’s CosmosDB née DocumentDB. An interesting observation is that even more sharding options are present which affect how well data can be managed by cloud-native applications. Interestingly, Aurora despite being a cloud service does not offer sharding for horizontal scalability. More variety is available at other providers, for instance Azure offering key-sharded data in CosmosDB which would otherwise resemble Cloudant.

4.1 Database Performance

The first experiment compares the deviation of response times as measure of instability between a local database system and a database system or service in the cloud, represented by AWS. A complex document management transaction consisting of six individual queries was performed with MySQL first as this system is reflected in the largest variety of cloud hosting options. The benchmark itself ran both on the local machine and as close as possible to the database, i.e. with high affinity in the cloud. Figure 6a shows that the local queries are much faster and their response time more predictable than those of the cloud counterpart when the benchmark runs locally and thus all queries need to traverse the wide-area network. Figure 6b contrasts the results with the affine benchmark. All such measurements are suffixed with */in-cloud*. The trivial comparison shows that a local benchmark with a local MySQL system performs equal to a Kubernetes-hosted benchmark and MySQL container pair, as both communicate via local link. As soon as the provider’s services are involved, this translates into a local-area network transmission within one hosting region.

In Fig. 7a, a different set of queries was tested with MongoDB, hence the different absolute times and network delay effects. Nevertheless, the cloud-hosted database container shows a higher stability in response times with both local and cloud benchmark, while the latter also has a lower response time as expected from the observation of MySQL. The interesting difference is that the response time deviations are high for local MongoDB queries but low for local MySQL queries which suggests that not only the network influences the variation in response times. In contrast, Fig. 7b reports on the same experiments using the MongoDB adapter for CosmosDB which was conducted over two non-consecutive days.

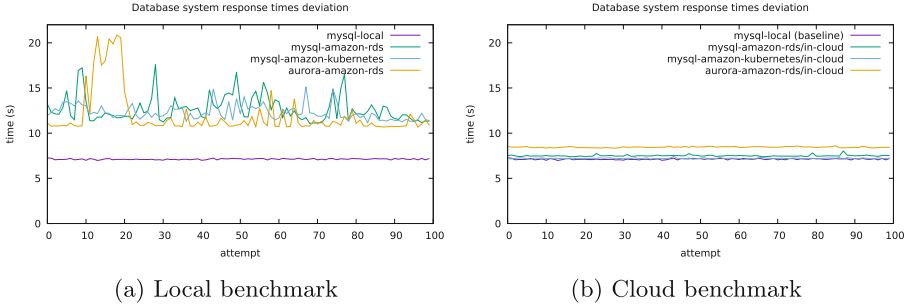


Fig. 6. Query times for MySQL

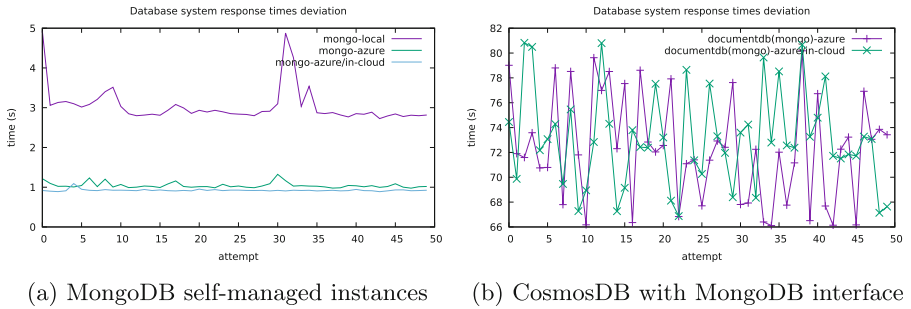


Fig. 7. Query times for MongoDB/CosmosDB, both local and cloud benchmarks

In both the local and cloud-hosted benchmark cases, the latter using an Azure VM, the performance is relatively stable within one day, varying a lot between the days (about 33%), and extremely low compared to the native MongoDB counterparts. Additionally, Fig. 8 compares two database services from Bluemix to complete the variations in engines, providers, services and benchmark locations. The interesting observation is that not only are the absolute response times of PostgreSQL strictly below the ones of MySQL ($\bar{r}t = 0.92$ vs. 6.23), their deviation is also a lot smaller ($\sigma = 2.60$ vs. 28.32).

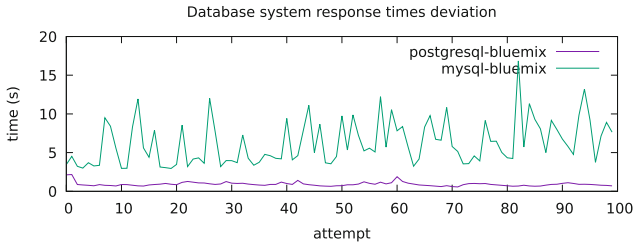


Fig. 8. Query times for MySQL and PostgreSQL, local benchmark

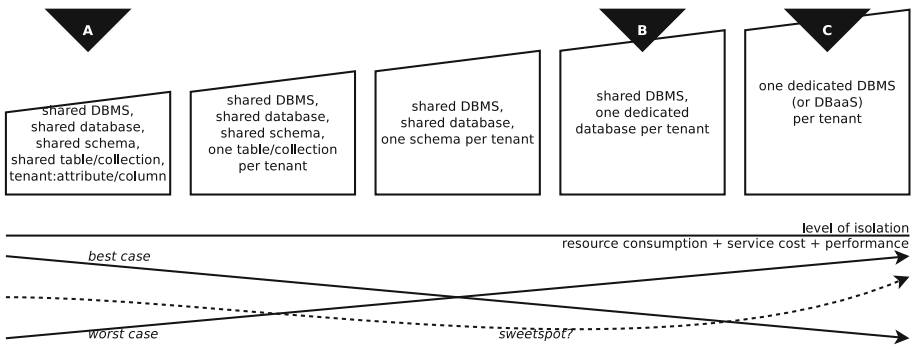


Fig. 9. Model of flexible multi-tenancy configurations for database services, with MongoDB

4.2 Database Multi-tenancy

Data management is affected by the level of isolation between the tenants in a multi-tenant database service setup. Figure 9 represents the model of matching isolation level to estimated performance and cost. For three out of the five different levels, we have measured the actual behaviour with three different implementations each.

Figure 10 contains the corresponding results. The multi-threaded implementation (MT) takes longer per thread to return the results but all threads return close to each other, leading to a speedup of 22.5%, 41.9% and 59.6% over the single-threaded implementation of A, B and C, respectively. Option C is the fastest and most isolated option, but does not represent an unconditional overall sweetspot due to also being the most expensive one.

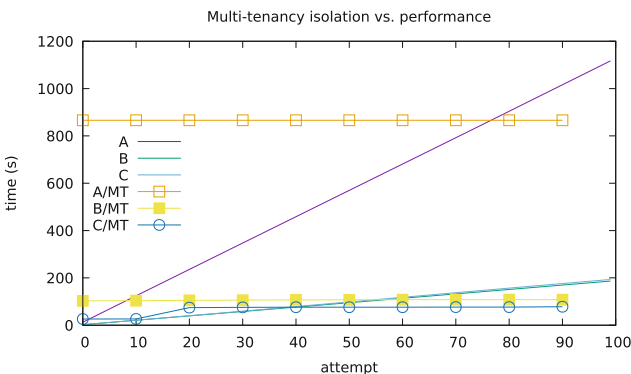


Fig. 10. Results for multi-tenancy options A, B and C with and without multi-threading

4.3 Database Pricing

Of interest to the application provider is the total cost of provisioning in relation to a quality of experience which allows for a surplus-generating revenue. Our findings indicate that there is no clear price advantage of self-managed containers on the SaaS level versus a comparable DBaaS option, or vice-versa, when taking replicated containers for higher resilience into account. From a methodic point of view, we derive an unquantified graphical representation of pricing in relation to performance, availability/resilience, reliability, multi-tenancy and scalability as shown in Fig. 11 and propose to derive a comparison tool for application engineers.

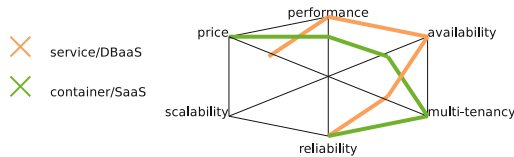


Fig. 11. Spider graph for pricing trade-offs, sampled for MySQL at Google; outside = best

5 Findings and Recommendations

As the selected results have shown, a general statement about a single best database option will not be possible, and a sharp definition of CNDB remains impossible. Our general recommendation is therefore that tools such as CNDBbench should be used in cloud application migration projects to produce metrics upon which selection decisions can be based.

Several systems and services have undocumented or undiscoverable limitations which can be revealed by systematic testing as is the case with CNDBbench. For instance, Crate only returns up to 10,000 rows by default and requires a LIMIT clause to return more. Azure CosmosDB limits the maximum requests to 1000 per second, which can be increased to 10,000, and requires the activation of further instances to grow beyond, despite low load on the database. Several protocols and client-side libraries are setting up timeouts. Some are merely difficult to deactivate, others even impossible, like the 20 s query timeout when inserting many records through PyMongo.

For the mentioned limitations, we recommend a discoverable description of these properties in addition to more complete documentation [6]. For the construction of future applications, assuming more maturity and choice in containerised database systems, we recommend auto-clustering microservices as currently implemented for Crate. In any case, the economics of self-managed instances depends to a large degree on the business background, including the skills and qualifications of the application engineers. In tech-savvy companies, self-managed database containers running on top of virtual machines using container management frameworks are recommended.

6 Discussion and Conclusion

We discuss our findings in the context of recent publications about both cloud-native databases and database characteristics in the cloud in general.

Szczyrbowski and Myszor present a behaviour comparison between the Oracle Database Schema Service which offers an HTTP interface [13] and the local 11g equivalent. Their main focus is on performance stability, minimising deviations in query times for three operations: INSERT, UPDATE and SELECT. Their approach is comparable to ours apart from updates and technological choices. The findings suggest that the cloud service has a much lower deviation apart from also being (presumably due to opaque hardware differences) faster in the worst, average and best case. We were able to reproduce this for MongoDB but not for MySQL, and therefore assume that their findings cannot be generalised.

Another performance comparison is authored by Seriatos et al. [12]. The focus is on three database systems – MongoDB, Cassandra and HBase – in the BONFIRE cloud testbed. Cost and scaling are not discussed. The YCSB benchmark is used. The findings tell that each of the system performs differently depending on the workload which implies two future work directions: The first, mentioned by the authors, is the tuning of parameters; the second, added by us, is the design of adaptive multi-database connectivity as the next evolutionary step for CNDBs.

The focus on cost is set by Mian et al. in an analysis of resource configuration using the TPC-C/E/H benchmarks in three application scenarios [9]. While the authors focus on AWS EC2, the DBaaS services of the same provider are not considered. A similar aim is conveyed in the work by Floratou et al. albeit with a critical look at unpleasant surprises in terms of financial risks when using DBaaS [5]. The findings are that more expensive hourly services may turn out more cost-effective overall, which is substantiated with observations of MySQL and SQL Server running on local hardware. The authors propose a benchmark-as-a-service for application developers (as database users). To cover the scaling and resilience characteristics which are important in a cloud setting, Bagui et al. look at sharding techniques and propose an implementation [1]. The work is demonstrated with MySQL and extends to other engines. Costa et al. examined partial database migration to the cloud [3]. The migration path in this work is from local PostgreSQL to AWS DynamoDB without giving up the former by adding a transparent adapter to the application. The finding is that scalability bottlenecks can be circumvented by offloading data to DynamoDB. While we have not analysed the same system, our results with non-ACID confirm this observation.

Table 4 summarises which of the cloud database properties were covered by related works and whether our findings agree (☑) or disagree (⊗) with them. When the results are not clear, the need for future experimental research (⚙️) is shown instead. The lack of a reusable testbed from the related work is evident.

We conclude that cloud-native databases are a challenging topic in need of more formal expressions concerning their configuration and characteristics and of

Table 4. Related work comparison

| Study | Performance | Scalability | Resilience | Tenancy | Price | Testbed |
|--------------------------|-------------|-------------|------------|---------|-------|---------|
| Szczyrbowski et al. [13] | ⊕ | | | | | |
| Seriatos et al. [12] | ⚙️ | | | | | |
| Mian et al. [9] | | | | | ⚙️ | |
| Floratou et al. [5] | | | | | ⚙️ | |
| Bagui et al. [1] | | ⚙️ | ⚙️ | | | |
| Costa et al. [3] | | ✓ | | | | |

more experiments. We suggest that future research should be directed towards a holistic approach of assessing flexible database options in the cloud which involve self-hosted data containers, blob storage services and DBaaS.

Repeatability

Our benchmark implementation, CNDBbench, is publicly available to repeat our experiments. For reference and reproducibility of the results, the experiment setup including hardware specifications and instructions is given in detail in a raw open science notebook which is made available together with a technical appendix due to the page number limitation. The notebook also contains reference results, additional experiments and findings concerning resilience, scalability and pricing^{1,2}. We encourage the critical examination and re-use of the datasets.

Acknowledgements. This research has been funded by the Swiss Commission for Technology and Innovation (CTI) in project ARKIS/18992.1. It has also been supported by an AWS in Education Research Grant, an IBM Academic Initiative for Cloud offer, a Microsoft Azure Research Award and a Google Cloud credit, all of which helped us to conduct our experiments on public commercial cloud environments.

References

1. Bagui, S., Nguyen, L.T.: Database sharding: to provide fault tolerance and scalability of big data on the cloud. *Int. J. Cloud Appl. Comput. (IJCAC)* **5**(2), 36–52 (2015)
2. Brunner, S., Blöchlinger, M., Toffetti, G., Spillner, J., Bohnert, T.M.: Experimental evaluation of the cloud-native application design. In: 4th International Workshop on Clouds and (eScience) Applications Management (CloudAM), Limassol, Cyprus, December 2015
3. Costa, C.H., Maia, P.H.M., Mendonça, N.C., Rocha, L.S.: Supporting partial database migration to the cloud using non-intrusive software adaptations: an experience report. In: Celesti, A., Leitner, P. (eds.) *ESOCC Workshops 2015*. CCIS, vol. 567, pp. 238–248. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_18

¹ CNDBbench: <https://github.com/serviceprototypinglab/cndbbench>.

² CNDBresults: <https://github.com/serviceprototypinglab/cndbresults>.

4. Costa, C.M., Leite, C.R.M., Sousa, A.L.: Efficient SQL adaptive query processing in cloud databases systems. In: IEEE EAIS, pp. 114–121, Natal, Brazil, May 2016
5. Floratou, A., Patel, J.M., Lang, W., Halverson, A.: When free is not really free: what does it cost to run a database workload in the cloud? In: Nambiar, R., Poess, M. (eds.) TPCTC 2011. LNCS, vol. 7144, pp. 163–179. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32627-1_12
6. Frey, S., Hasselbring, W., Schnoor, B.: Automatic conformance checking for migrating software systems to cloud infrastructures and platforms. *J. Softw. Evol. Proc.* **25**(10), 1089–1115 (2013)
7. Goldschmidt, T., Jansen, A., Koziolok, H., Doppelhamer, J., Breivold, H.P.: Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In: 7th IEEE International Conference on Cloud Computing (CLOUD). pp. 602–609, Anchorage, Alaska, USA, July 2014
8. Götz, S., Ilsche, T., Cardoso, J., Spillner, J., Kissinger, T., Aßmann, U., Lehner, W., Nagel, W.E., Schill, A.: Energy-efficient databases using sweet spot frequencies. In: 1st International Workshop on Green Cloud Computing (GCC), pp. 871–876, London, UK, December 2014
9. Mian, R., Martin, P., Zulkernine, F.H., Vázquez-Poletti, J.L.: Cost-effective resource configurations for multi-tenant database systems in public clouds. *Int. J. Cloud Appl. Computing (IJCAC)* **5**(2), 1–22 (2015)
10. Nguyen, H., Shen, Z., Gu, X., Subbiah, S., Wilkes, J.: AGILE: elastic distributed resource scaling for infrastructure-as-a-service. In: 10th International Conference on Autonomic Computing (ICAC), San Jose, California, USA, pp. 69–82, June 2013
11. Sakr, S.: Cloud-hosted databases: technologies, challenges and opportunities. *Cluster Comput.* **17**(2), 487–502 (2014)
12. Seriatos, G., Kousiouris, G., Menychtas, A., Kyriazis, D., Varvarigou, T.: Comparison of database and workload types performance in cloud environments. In: Karydis, I., Sioutas, S., Triantafyllou, P., Tsoumakos, D. (eds.) ALGO CLOUD 2015. LNCS, vol. 9511, pp. 138–150. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29919-8_11
13. Szczyrbowski, M., Myszor, D.: Comparison of the behaviour of local databases and databases located in the cloud. In: Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D. (eds.) BDAS 2015-2016. CCIS, vol. 613, pp. 253–261. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34099-9_19
14. Wiese, L.: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases. DeGruyter/Oldenbourg, Berlin (2015)



Testing and Comparing the Performance of Cloud Service Providers Using a Service Broker Architecture

Divyaa Manimaran Elango¹, Frank Fowley¹, and Claus Pahl²(✉)

¹ IC4, Dublin City University, Dublin, Ireland

² SwSE, Free University of Bozen-Bolzano, Bolzano, Italy
Claus.Pahl@unibz.it

Abstract. Service brokers are tools that allow different individual service providers to be integrated. An API can be a mechanism to provide a joint interface. Broker can actually also be use for more than integration. We use a cloud service broker that implements a multi-cloud abstraction API in order to carry out performance comparisons between different cloud services. The broker tool here is a multi-cloud storage API that integrates a number of provided storage services. The library supporting the API is organised into three services, which are a file, a blob and a table service. Using this broker architecture, we developed a performance test scenario to compare the different providers, i.e., to compare a range of storage operations by different providers.

1 Introduction

Integration is a key problem in the cloud services context. A cloud service broker is an intermediary application between a client and cloud provider service that can provide this integration [15]. Brokerage reduces the need for service consumers to analyze different types of services by different providers [1]. This enables a single platform to offer the client a common cloud storage service. This results in cost optimization and reduced level of back-end data management requirements.

For our performance evaluation, we use here a cloud service broker that implements a multi-cloud abstraction API. This multi-cloud storage broker supports GoogleDrive, DropBox, Microsoft Azure and Amazon Web Services as the provided storage services. The API library offers file, blob and table services. The API can facilitates the distribution of different types of cloud provider services [16]. The abstraction library allows the cloud broker to adapt to a rapidly changing marketplace.

Vendor lock-in is often referred to as a critical point in choosing a provider. In order to avoid lock-in, a broker can help. A multi-cloud abstraction library is a suitable mechanism that it makes it easy for the client to switch between cloud providers with different services that are supported by the broker.

Switching or migrating between providers can be driven by quality [27,32]. We use a broker implementation to compare the supported services [8,13,23] from a performance perspective [42]. Service brokers normally remedy interoperability problems [2–5,10]. However, based on this architecture, we look into other service qualities, namely performance which is of key importance for all cloud layers [28–30]. A performance test application was developed here to compare between the services provided through the broker [19]. The performance test scenario was used to compare a range storage operations across different supported providers.

2 Broker – Principles and Supported Services

In Fig. 1 we have outlined the core components of the broker architecture. We also discuss the storage services supported by it in this section.

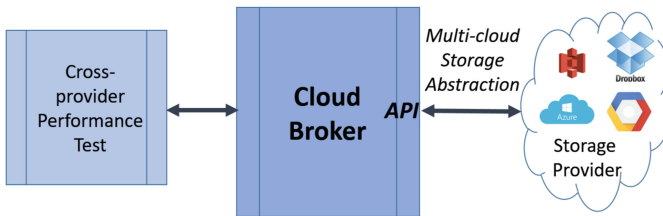


Fig. 1. Architecture of a multi-cloud storage broker.

2.1 Principle Properties of the Storage Broker

Cloud services are generally provided with specifications, but often constructed in a way that makes them hard to be used as part of a common interface, thus impeding on interoperability. We studied different multi-cloud libraries, including Apache Jclouds, DeltaCloud, Kloudless, SecureBlackBox, and SimpleCloud. The purpose was to adopt a successful solution template.

We decided to adopt an approach similar to the Apache Jclouds library for abstraction, as we will explain now. Jclouds [22] provides cloud-agnostic abstraction. A single instance context approach for the mapping of a user request in jclouds was used in our implementation. The purpose of having each class for each provider across different levels of service was adopted from a similar design in the SecureBlackBox library. Our concept of including a manager interface layer at each component level is adopted from LibCloud, another library.

2.2 Services Supported by the Broker

We have included storage services from Google, Dropbox, Azure and Amazon in our cloud storage broker.

- *Amazon Web Service S3* [35] is a file storage service which is built on REST and SOAP. Their SDK is available in all major development languages.
- *Azure Storage* [37] supports blob, file, queue and table services. The API is built on REST, HTTP and xml, and can be integrated with Microsoft visual studio, eclipse and GIT. The Azure SDK provides a separate API package for each service and has the same code flow across different service APIs.
- *DropBox* [36] is a file hosting service. It also enables synchronised backup and web sharing. The DropBox API is very light-weight and easy for a new user.
- *GoogleDrive* [38] offers a cloud file storage service. The GoogleDrive service includes access to a Google API client library.

This selection of service providers resulted in a grouping of the cloud providers and their services as shown in the table below¹ that summarises the main features of the services:

| Service | Azure | AWS | Google | DropBox |
|---------|---------------------------|--------------------|-------------|---------|
| File | Storage file | - | GoogleDrive | DropBox |
| Blob | Storage blob | AWS S3 | - | - |
| Table | Storage Table, DocumentDB | DynamoDB, SimpleDB | - | - |

3 Broker Architecture

Portability and interoperability are the key objectives of a cloud brokerage tool. Thus the objective of designing and developing an abstraction API is to produce an effective cross-service cloud delivery model [14]. Before describing how we use this to support performance evaluation, we still need to introduce the architectural principles. The main service-oriented functionalities of cloud providers are compute nodes, data volume, load balance, DNS and so on. The advantage of bringing these functionalities to a multi-cloud application provides (1) an easy way of importing and exporting data, (2) choice over price, (3) enhanced SLA, and (4) the elimination of vendor lock-in.

As concept and function integration is the key difficulty in constructing the broker, this broker implementation is based on an ontology that at conceptual level defines the integration. This Storage Abstraction Ontology describes the common naming and meaning approach of the abstraction API [25, 26, 34]. The model consists of four main layers, namely Service, Provider, Level-2 (composite storage objects) and Level-1 (core storage objects).

- *Level-4 Service*: The Service layer is the top layer and is directly integrated to the user interface layer. This layer basically describes the services that the multi-cloud storage abstraction API supports. There are three services currently supported. They are Blob, Table and File service.
- *Level-3 Provider*: The Provider layer is the second layer, which is one of the context object parameters mapped to the service layer. The multi-cloud storage abstraction supports four main providers, namely Microsoft Azure,

¹ [https://www.google.com/drive/;](https://www.google.com/drive/)
[https://www.dropbox.com/;](https://www.dropbox.com/)
[https://aws.amazon.com/s3/;](https://aws.amazon.com/s3/)
[https://azure.microsoft.com/en-us/services/storage/;](https://azure.microsoft.com/en-us/services/storage/)

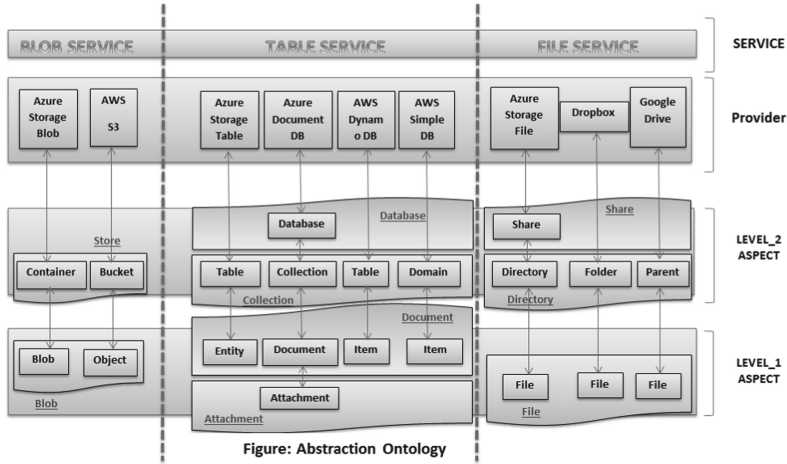


Fig. 2. Ontology-based layered broker architecture.

Amazon Web Services, GoogleDrive and DropBox. The corresponding services supported by the providers are shown below:

| Service | Provider |
|---------|---|
| Blob | Azure storage blob; AWS S3 |
| Table | Azure storage table; Azure DocumentDB; AWS DynamoDB; AWS SimpleDB |
| File | Azure storage file; DropBox and GoogleDrive |

- *Level-2 Composite:* Level-2 is the next layer. This layer represents the first level or higher level of composite object abstraction. This layer is service-neutral and brings out the common naming across the providers specific functionalities. Each layer is abstracted based on the common operations and aspect of how the main function is applied in that particular service. Common naming is represented to easily categorise storage resources and group them to make the development of the coding easier.

Based on the Abstraction Ontology Fig. 2, the Blob service has Store which groups Container from Azure Storage Blob and Bucket from AWS S3. The Table service has two different sub-layers - where Database belongs to Azure DocumentDB Database, and where Collection groups Table from Azure Storage Table, collection from Azure DocumentDB Collection, Table from AWS DynamoDB and Domain from AWS SimpleDB. The File service has two different sub-layers where Share belongs to Azure Storage File and Directory groups Directory from Azure Storage File, Folder from DropBox and Parent from the GoogleDrive service.

- *Level-1 Core:* Level-1 represents the lower level of core object abstraction. This layer contains the core functionalities of a particular service across different providers. The classes in this level are extended from an abstract class called AbstractConnector. The class implements the abstract methods defined in the AbstractConnector class. The mapping from Level-2 to Level-1

is performed by an interface class called *Manager*. This *Manager* identifies the provider class by its key. Basic CRUD operations on the storage resources are included as core methods. In order to achieve these functions, each operation request should pass through the Level-2 mappings and are then mapped across the service and providers.

The *Blob* service has *Blob* which groups *Blob* from Azure Storage *Blob* and *Object* from AWS *S3*. The *Table* service has two different sub layers. It has *Item* which groups *Entity* from Azure Storage *Table*, *Document* from Azure *DocumentDB* *Document*, *Item* from AWS *DynamoDB* and *Item* from AWS *SimpleDB*. Also, the second sub layer *Attachment* belongs to Azure *DocumentDB*. The *File* service has *File* which groups *File* from Azure Storage *File*, *File* from *DropBox* and *File* from *GoogleDrive*.

4 Performance Testing and Provider Comparison

We have used the broker to compare performance values for the four providers selected. The broker is instrumented to provide the response time results.

In this section, we describe the performance test set-up and the results for the three service types *blob*, *file* and *table* across the different providers. We organise this section based on the storage types *blob*, *file* and *table*.

Not all providers support each of the storage types. So, the number of compared services provided varies between two and four. We report on the time consumed for a number of standard operations at the two important levels 1 and 2 of the layer architecture. In this way, we cover individual objects (*items*) and composites (*collections*) and a range of standard operations on them such as creating or deleting.

4.1 Blob Service Performance Test

The *Blob Service Performance Test* was performed on two providers, namely Azure Storage *Blob* and AWS *S3*. This performance test includes two object levels. Level-2 represents *Store* (which includes *container* and *Bucket*). Level-1 represents *Blob* (which includes *Blob* and *Object*).

- The total number of tests performed was 27 to fully cover the respective core and composite objects and the different relevant operations on them. The performance test compares the operations across the service providers.
- Each operation was run 10 times in order to avoid any accidental performance irregularities due to external factors, and the corresponding process time for each request from T1 to T10 was calculated.
- Each request was processed with the same blob size of 10.2 MB, which resembles a standard object size.

The result includes start time, end time, average time and total duration – see Figs. 3 and 4.

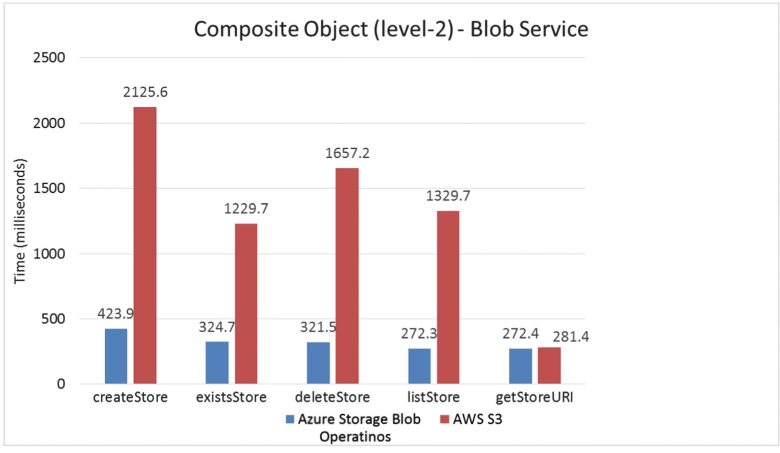


Fig. 3. Blob service Level-2 composite object.

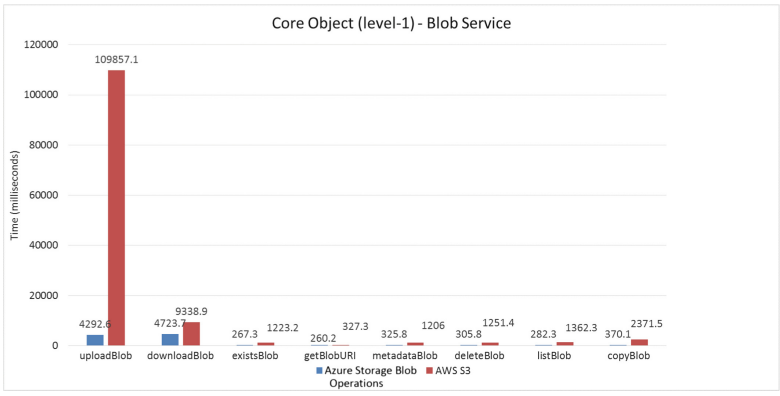


Fig. 4. Blob service Level-1 core object.

4.2 File Service Performance Test

The *File Service Performance Tests* were performed on Azure Storage File, GoogleDrive and DropBox. The tests include two object levels. Level-2 represents Share and Directory. Level-1 represents Files.

- The total number of tests performed was 26 to cover the combinations of different object types and different operations on them.
- The performance tests compare the operations across the service providers. Each operation was run 10 times to eliminate irregular single behaviour, and the corresponding process time for each request from T1 to T10 was calculated.
- Each request was processed with same file size of 10.2 MB as a common size for the object type in question.

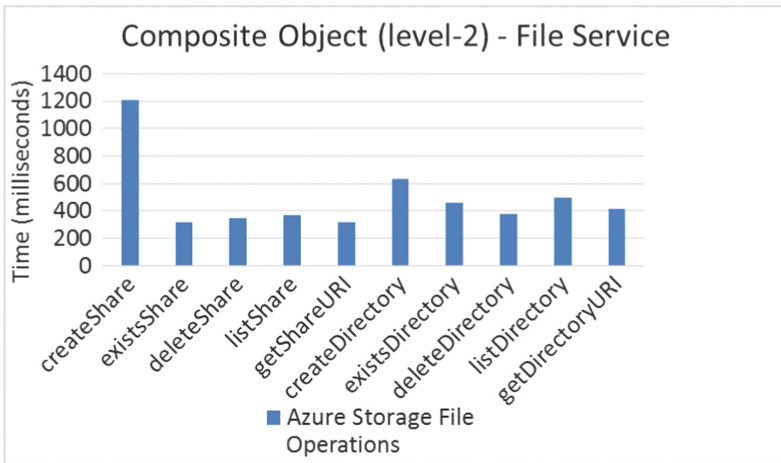


Fig. 5. File service Level-2 composite object.

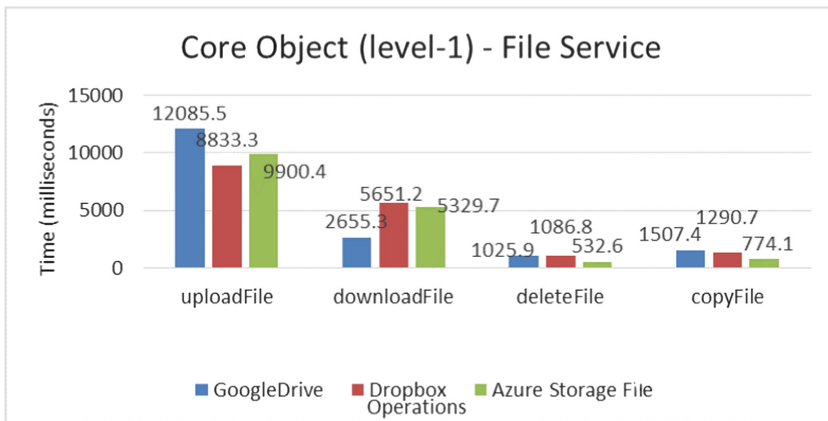


Fig. 6. File service Level-1 core object.

The results include start time, end time, average time and total duration – see Figs. 5 and 6.

4.3 Table Service Performance Test

The *Table Service Performance Tests* were performed on Azure Storage Table, Azure DocumentDB, AWS DynamoDB and AWS SimpleDB. The Tests include two object levels. Level-2 represents Database and Collections (which includes Table, Collections, Table and Domain). Level-1 represents Item (which includes Entity, Document, Table Item, Domain Item) and Attachment.

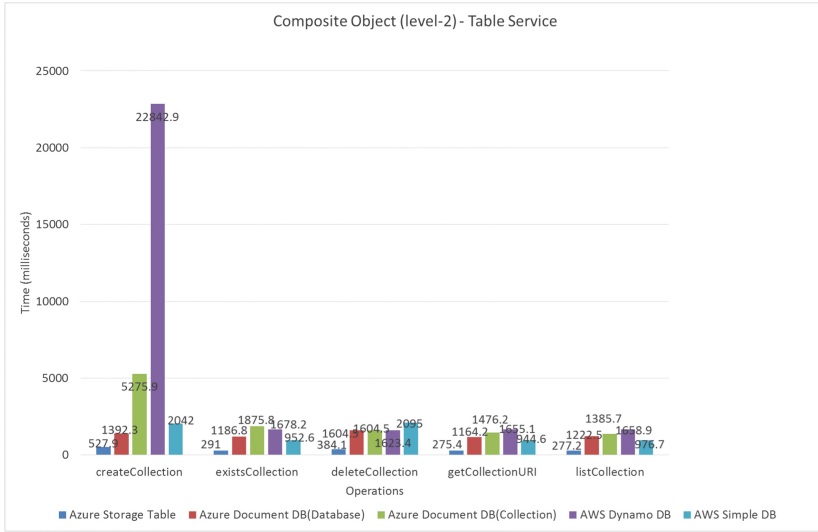


Fig. 7. Table service Level-2 composite object.

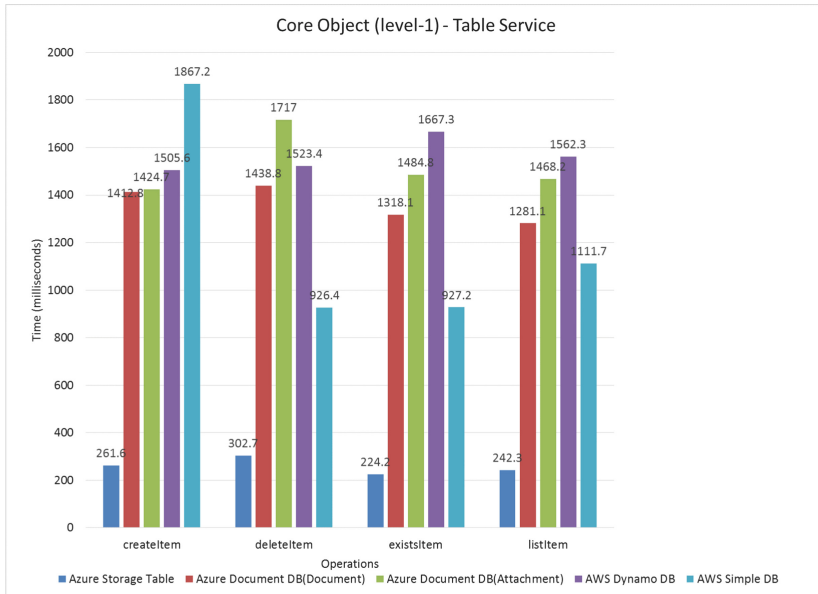


Fig. 8. Table service Level-1 core object.

- The total number of tests was 45. As already explained for the blob tests, this number covers the combination of different objects and the different operations on them. The performance tests compare the operations then across the service providers.

– Each operation was run 10 times, as earlier to avoid irregularities, and the corresponding process time for each request from T1 to T10 was calculated. Each request was processed with a single data record of approximately four columns.

The results include start time, end time, average time, average time and total duration – see Figs. 7 and 8 (and also the performance details in Fig. 9).

| SERVICE | PROVIDER | LEVEL OF ASPECTS | Operation Type | Parameter: Inputs | Filesize | Start Time | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | End Time | Avg Time | Duration | Avg | |
|--------------|------------------------|------------------|-----------------|----------------------------|----------|------------|--------|--------|-------|--------|--------|--------|--------|--------|----------|----------|----------|----------|----------|----------|-------|
| BlobService | Azure Storage Blob | LEVEL_2 | existsContainer | mainName=Naostocant... | 10.2 MB | 1478.612 | 3004 | 105 | 114 | 117 | 112 | 116 | 118 | 138 | 111 | 184 | 138 | 1478.612 | 3004 | 00:00:03 | 423.9 |
| BlobService | Azure Storage Blob | LEVEL_2 | deleteContainer | mainName=Naostocant... | 10.2 MB | 1478.614 | 2575 | 78 | 71 | 70 | 69 | 68 | 69 | 71 | 70 | 74 | 1478.614 | 2575 | 00:00:03 | 321.5 | |
| BlobService | Azure Storage Blob | LEVEL_2 | listContainer | | 10.2 MB | 1478.613 | 2153 | 73 | 60 | 56 | 65 | 60 | 56 | 59 | 76 | 65 | 1478.613 | 2153 | 00:00:02 | 272.3 | |
| BlobService | Azure Storage Blob | LEVEL_2 | getContainerURL | mainName=Naostocant... | 10.2 MB | 1478.613 | 2388 | 88 | 38 | 35 | 41 | 35 | 40 | 39 | 47 | 43 | 1478.613 | 2388 | 00:00:02 | 272.4 | |
| BlobService | AWS S3 | LEVEL_2 | existsContainer | mainName=Naostocan... | 10.2 MB | 1478.619 | 6381 | 1828 | 1711 | 1711 | 1712 | 1712 | 1712 | 1712 | 1712 | 1712 | 1478.619 | 6381 | 00:00:09 | 225.8 | |
| BlobService | AWS S3 | LEVEL_2 | deleteContainer | mainName=Naostocan... | 10.2 MB | 1478.706 | 5394 | 1186 | 1071 | 1173 | 1045 | 1152 | 1188 | 1350 | 1167 | 1046 | 1478.706 | 5394 | 00:00:16 | 1657.2 | |
| BlobService | AWS S3 | LEVEL_2 | listContainer | | 10.2 MB | 1478.705 | 226 | 128 | 141 | 129 | 120 | 119 | 120 | 147 | 148 | 147 | 1478.705 | 226 | 00:00:01 | 376.1 | |
| BlobService | AWS S3 | LEVEL_2 | getContainerURL | mainName=Naostocan... | 10.2 MB | 1478.7 | 2649 | 10 | 17 | 15 | 16 | 15 | 17 | 17 | 14 | 14 | 1478.7 | 2649 | 00:00:01 | 281.4 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Upload | url=filePath(Project) | 10.2 MB | 1478.614 | 9781 | 3553 | 5137 | 3245 | 3110 | 3141 | 2914 | 4522 | 4103 | 3420 | 1478.614 | 4292 | 00:00:21 | 4292.6 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Download | url=filePath(View) | 10.2 MB | 1478.614 | 7225 | 5097 | 4904 | 4083 | 4576 | 4215 | 4151 | 4117 | 4139 | 4450 | 1478.614 | 4723 | 00:00:17 | 4723.7 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Exists | url=BlobContainerName | 10.2 MB | 1478.614 | 2023 | 66 | 63 | 68 | 76 | 68 | 67 | 71 | 70 | 74 | 1478.614 | 2023 | 00:00:01 | 267.3 | |
| BlobService | Azure Storage Blob | LEVEL_1 | URL | url=BlobContainerName | 10.2 MB | 1478.614 | 2277 | 37 | 35 | 33 | 34 | 33 | 41 | 32 | 46 | 34 | 1478.614 | 2277 | 00:00:01 | 260.2 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Metadata | url=BlobContainerName | 10.2 MB | 1478.615 | 2879 | 44 | 34 | 45 | 43 | 41 | 39 | 38 | 37 | 38 | 1478.615 | 2879 | 00:00:01 | 325.8 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Delete | url=BlobContainerName | 10.2 MB | 1478.616 | 2385 | 78 | 81 | 74 | 66 | 67 | 71 | 70 | 73 | 95 | 1478.616 | 2385 | 00:00:01 | 305.8 | |
| BlobService | Azure Storage Blob | LEVEL_1 | List | url=mainName=Naostocant... | 10.2 MB | 1478.615 | 2135 | 66 | 75 | 75 | 75 | 75 | 77 | 91 | 73 | 73 | 1478.615 | 2135 | 00:00:01 | 282.3 | |
| BlobService | Azure Storage Blob | LEVEL_1 | ListBatch | url=mainName=Naostocant... | 10.2 MB | 1478.615 | 2189 | 91 | 100 | 85 | 77 | 77 | 76 | 75 | 76 | 81 | 1478.615 | 2189 | 00:00:01 | 292.7 | |
| BlobService | Azure Storage Blob | LEVEL_1 | Copy | url=DestinationContainer | 10.2 MB | 1478.616 | 2251 | 136 | 234 | 160 | 179 | 202 | 174 | 147 | 148 | 147 | 1478.616 | 2251 | 00:00:01 | 376.1 | |
| BlobService | AWS S3 | LEVEL_1 | Upload | url=filePath(Project) | 10.2 MB | 1478.702 | 115172 | 113111 | 99692 | 106843 | 105346 | 107147 | 109442 | 112648 | 112116 | 108004 | 1478.702 | 109587 | 00:00:18 | 10957.1 | |
| BlobService | AWS S3 | LEVEL_1 | Download | url=filePath(Project) | 10.2 MB | 1478.704 | 18565 | 7761 | 10831 | 7672 | 7607 | 7502 | 7873 | 8285 | 7968 | 8773 | 1478.704 | 9338 | 00:00:13 | 9338.9 | |
| BlobService | AWS S3 | LEVEL_1 | Exists | url=BlobRefToBucket | 10.2 MB | 1478.704 | 4482 | 614 | 784 | 773 | 1449 | 798 | 809 | 854 | 797 | 760 | 1478.704 | 4482 | 00:00:12 | 3223.2 | |
| BlobService | AWS S3 | LEVEL_1 | URL | url=BlobRefToBucket | 10.2 MB | 1478.704 | 3061 | 25 | 20 | 21 | 30 | 21 | 25 | 21 | 19 | 1478.704 | 3061 | 00:00:03 | 327.3 | | |
| BlobService | AWS S3 | LEVEL_1 | Metadata | url=BlobRefToBucket | 10.2 MB | 1478.705 | 4758 | 844 | 1043 | 818 | 813 | 787 | 722 | 816 | 755 | 704 | 1478.705 | 4758 | 00:00:12 | 1206 | |
| BlobService | AWS S3 | LEVEL_1 | Delete | url=BlobRefToBucket | 10.2 MB | 1478.705 | 5307 | 740 | 697 | 730 | 707 | 841 | 1279 | 790 | 740 | 762 | 1478.705 | 5307 | 00:00:11 | 1253.4 | |
| BlobService | AWS S3 | LEVEL_1 | List | url=mainName=Naostocan... | 10.2 MB | 1478.705 | 5284 | 1018 | 912 | 899 | 1013 | 888 | 880 | 897 | 818 | 919 | 1478.705 | 5284 | 00:00:01 | 1362.3 | |
| BlobService | AWS S3 | LEVEL_1 | Copy | url=DestinationContainer | 10.2 MB | 1478.705 | 6883 | 1904 | 1756 | 1834 | 2032 | 2087 | 1829 | 1875 | 1781 | 1734 | 1478.705 | 6883 | 00:00:21 | 2371.5 | |
| FileService | Azure Storage File | LEVEL_2 | createShare | url=Name=TestShare | 10.2 MB | 1478.523 | 5052 | 985 | 916 | 770 | 878 | 741 | 562 | 683 | 768 | 728 | 1478.523 | 5052 | 00:00:12 | 1208.3 | |
| FileService | Azure Storage File | LEVEL_2 | deleteShare | url=Name=TestShare | 10.2 MB | 1478.529 | 4482 | 614 | 784 | 773 | 1449 | 798 | 809 | 854 | 797 | 760 | 1478.529 | 4482 | 00:00:12 | 1208.3 | |
| FileService | Azure Storage File | LEVEL_2 | listShare | | 10.2 MB | 1478.529 | 2744 | 72 | 71 | 69 | 70 | 70 | 74 | 78 | 100 | 106 | 1478.529 | 2744 | 00:00:03 | 347.4 | |
| FileService | Azure Storage File | LEVEL_2 | existsShare | url=Name=TestShare | 10.2 MB | 1478.532 | 2990 | 123 | 112 | 54 | 63 | 84 | 75 | 77 | 53 | 76 | 1478.532 | 2990 | 00:00:03 | 370.7 | |
| FileService | Azure Storage File | LEVEL_2 | getShareURL | url=Name=TestShare | 10.2 MB | 1478.532 | 2281 | 61 | 68 | 61 | 51 | 51 | 51 | 51 | 51 | 51 | 1478.532 | 2281 | 00:00:03 | 405.0 | |
| FileService | Azure Storage File | LEVEL_2 | createDirectory | url=Name=TestDir | 10.2 MB | 1478.532 | 4317 | 130 | 303 | 108 | 178 | 145 | 139 | 189 | 445 | 162 | 1478.532 | 4317 | 00:00:06 | 635 | |
| FileService | Azure Storage File | LEVEL_2 | existsDirectory | url=Name=TestDir | 10.2 MB | 1478.532 | 3726 | 81 | 109 | 95 | 131 | 76 | 110 | 93 | 112 | 94 | 1478.532 | 3726 | 00:00:04 | 462.7 | |
| FileService | Azure Storage File | LEVEL_2 | deleteDirectory | url=Name=TestDir | 10.2 MB | 1478.539 | 2838 | 105 | 107 | 105 | 106 | 123 | 103 | 103 | 103 | 103 | 1478.539 | 2838 | 00:00:01 | 379.9 | |
| FileService | Azure Storage File | LEVEL_2 | listDirectory | url=Name=TestDir | 10.2 MB | 1478.534 | 2779 | 89 | 202 | 190 | 202 | 202 | 202 | 202 | 202 | 202 | 1478.534 | 2779 | 00:00:01 | 488.7 | |
| FileService | Azure Storage File | LEVEL_2 | getDirectoryURL | url=Name=TestDir | 10.2 MB | 1478.532 | 2481 | 205 | 204 | 205 | 205 | 205 | 205 | 205 | 205 | 205 | 1478.532 | 2481 | 00:00:01 | 415.1 | |
| FileService | Azure Storage File | LEVEL_1 | Upload | url=filePath(Project) | 10.2 MB | 1478.534 | 2474 | 4738 | 11320 | 14444 | 7663 | 10013 | 9900 | 6784 | 4713 | 3967 | 1478.534 | 2474 | 00:00:19 | 9000.4 | |
| FileService | Azure Storage File | LEVEL_1 | Download | url=filePath(Project) | 10.2 MB | 1478.534 | 6918 | 4804 | 4967 | 4950 | 4952 | 4952 | 4952 | 4952 | 4952 | 4952 | 1478.534 | 6918 | 00:00:12 | 1208.3 | |
| FileService | Azure Storage File | LEVEL_1 | Exists | url=Name=TestShare | 10.2 MB | 1478.536 | 2779 | 296 | 315 | 299 | 311 | 305 | 308 | 304 | 279 | 34 | 1478.536 | 2779 | 00:00:01 | 553 | |
| FileService | Azure Storage File | LEVEL_1 | Metadata | url=Name=TestShare | 10.2 MB | 1478.538 | 2728 | 157 | 131 | 110 | 157 | 134 | 146 | 103 | 110 | 131 | 1478.538 | 2728 | 00:00:01 | 390.7 | |
| FileService | Azure Storage File | LEVEL_1 | Delete | url=Name=TestShare | 10.2 MB | 1478.538 | 2728 | 157 | 131 | 110 | 157 | 134 | 146 | 103 | 110 | 131 | 1478.538 | 2728 | 00:00:01 | 390.7 | |
| FileService | Azure Storage File | LEVEL_1 | Copy | url=Name=TestShare | 10.2 MB | 1478.537 | 4283 | 352 | 307 | 254 | 295 | 384 | 627 | 603 | 280 | 316 | 1478.537 | 4283 | 00:00:17 | 774.1 | |
| FileService | Dropbox | LEVEL_1 | Upload | url=filePath(Project) | 10.2 MB | 1478.283 | 10746 | 9155 | 7385 | 8566 | 9436 | 10310 | 9354 | 8973 | 8459 | 8449 | 1478.283 | 8833 | 00:00:21 | 8833.3 | |
| FileService | Dropbox | LEVEL_1 | Delete | url=filePath(Project) | 10.2 MB | 1478.284 | 3330 | 501 | 503 | 557 | 530 | 482 | 482 | 471 | 506 | 520 | 1478.284 | 3330 | 00:00:01 | 786.2 | |
| FileService | Dropbox | LEVEL_1 | Download | url=filePath(Project) | 10.2 MB | 1478.518 | 6918 | 4804 | 4967 | 4950 | 4952 | 4952 | 4952 | 4952 | 4952 | 4952 | 1478.518 | 6918 | 00:00:12 | 1208.3 | |
| FileService | Dropbox | LEVEL_1 | Metadata | url=filePath(Project) | 10.2 MB | 1478.519 | 3385 | 722 | 666 | 726 | 707 | 1634 | 777 | 756 | 725 | 747 | 1478.519 | 3385 | 00:00:01 | 1068.8 | |
| FileService | Dropbox | LEVEL_1 | Copy | url=DestinationContainer | 10.2 MB | 1478.519 | 3957 | 810 | 864 | 818 | 820 | 1344 | 1134 | 767 | 659 | 1554 | 1478.519 | 3957 | 00:00:12 | 1208.3 | |
| FileService | GoogleDrive | LEVEL_1 | List | url=mainName=Naostocant... | 10.2 MB | 1478.28 | 2788 | 314 | 329 | 294 | 280 | 359 | 317 | 379 | 292 | 275 | 1478.28 | 2788 | 00:00:05 | 562.7 | |
| FileService | GoogleDrive | LEVEL_1 | Download | url=filePath(Project) | 10.2 MB | 1478.272 | 5484 | 2332 | 2303 | 2524 | 2475 | 2287 | 2423 | 2001 | 2868 | 1766 | 1478.272 | 2655 | 00:00:21 | 2655.3 | |
| FileService | GoogleDrive | LEVEL_1 | Delete | url=filePath(Project) | 10.2 MB | 1478.277 | 3892 | 614 | 784 | 773 | 1449 | 798 | 809 | 854 | 797 | 760 | 1478.277 | 3892 | 00:00:12 | 1208.3 | |
| FileService | GoogleDrive | LEVEL_1 | Copy | url=DestinationContainer | 10.2 MB | 1478.279 | 4400 | 1083 | 1240 | 1000 | 1101 | 1016 | 1119 | 1093 | 1163 | 1069 | 1478.279 | 4400 | 00:00:15 | 1507.4 | |
| TableService | Azure Storage Table | LEVEL_2 | createTable | url=Name=TestTable | 1478.736 | 2634 | 402 | 306 | 280 | 275 | 288 | 324 | 252 | 275 | 263 | 1478.736 | 2634 | 00:00:05 | 527.9 | | |
| TableService | Azure Storage Table | LEVEL_2 | existsTable | url=Name=TestTable | 1478.736 | 2470 | 49 | 49 | 44 | 44 | 40 | 42 | 49 | 44 | 49 | 1478.736 | 2470 | 00:00:01 | 291 | | |
| TableService | Azure Storage Table | LEVEL_2 | deleteTable | url=Name=TestTable | 1478.736 | 2987 | 107 | 103 | 88 | 92 | 90 | 86 | 88 | 92 | 1478.736 | 2987 | 00:00:01 | 384.7 | | | |
| TableService | Azure Storage Table | LEVEL_2 | getTableURL | url=Name=TestTable | 1478.736 | 2338 | 42 | 43 | 40 | 42 | 47 | 43 | 53 | 53 | 53 | 1478.736 | 2338 | 00:00:01 | 275.4 | | |
| TableService | Azure Storage Table | LEVEL_2 | listTable | | 1478.737 | 2372 | 48 | 43 | 51 | 45 | 42 | 47 | 42 | 42 | 43 | 1478.737 | 2372 | 00:00:01 | 277.2 | | |
| TableService | Azure Storage Document | LEVEL_2 | createDoc | url=Name=TestDoc | 1478.782 | 4091 | 1233 | 1121 | 1027 | 1115 | 1138 | | | | | | | | | | |

5 Discussion of Results and Conclusions

The aim of cloud service brokerage is customising or integrating existing services or making them interoperable. We have developed what based on common classification schemes in [11, 12, 39] is categorised as an integration broker. The purpose of a broker is intermediation between consumers and providers to provide advanced capabilities (interoperability and portability [33]) that builds up on an intermediary/broker platform to provide for instance a marketplace to bring providers and customers together and automatically facilitate multi-provider usage or portability across providers. The broker for cloud storage service providers implement a joint interface to allow

- easy portability for the user and
- easy extensibility for the broker provider.

This broker solution enables through the joint API also the opportunity for a cloud storage user to easily migrate between service providers and evolve the systems [9, 21], without having sufficient standards [6, 7, 20].

We investigated here the usage of the broker to carry out comparative performance tests across the providers in order to support the user with the decision which provider to choose, if this is taken based on a performance criterion.

Our observations from the performance tests we described earlier are the following:

- (a) Core Objects: We can observe that the performance of core object storage operations varies significantly across Cloud providers. Azure outperforms AWS S3 by a factor of between 4 and 5 in our test scenario. For individual object operations, Azure is also up to 5 times faster in terms of access speed. For example, the common function of UploadBlob takes approximately 4 seconds on Azure and 10 seconds on AWS S3 for a 10.2 MB file.
- (b) Composite Objects: The tests of composite object operations [31] that relate to collections show that Azure has significantly more access performance than other providers. In particular, AWS DynamoDB has a unusually long access time for its CollectionCreate operation. The tests on individual table entity operations show Azure to be the fastest by a considerable margin with over 5 to 6 times lesser access speeds on average.
- (c) Upload and Download: The average of the combined file upload and download speeds do not vary considerably across the providers tested.

We have defined some parameters, such as object size, in a specific way. Other choices might result in different observations. Our aim here was not to recommend a particular provider. The aim was to demonstrate the usefulness of instrumenting brokers for either decision making or as an ongoing monitoring approach. Any selection can anyway not happen without considering other properties such as security.

In the future, we plan to consider more storage services. Furthermore, the impact of different architectures in terms on IaaS or PaaS with and without the use of container technologies [17, 18, 24, 40, 41] shall be explored.

Acknowledgements. This work was partly supported by IC4 (Irish Centre for Cloud Computing and Commerce), funded by EI and the IDA.

References

1. Ried, S.: Cloud Broker - A New Business Model Paradigm. Forrester (2011)
2. Elango, D.M., Fowley, F., Pahl, C.: An ontology-based architecture for an adaptable cloud storage broker. In: *Advances in Service-Oriented and Cloud Computing*. Springer CCIS (2018, to appear)
3. Benslimane, D., Dustdar, S., Sheth, A.: Services mashups - the new generation of web applications. *Internet Comput.* **12**(5), 13–15 (2008)
4. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., Morrow, M.: Blueprint for the inter-cloud: protocols and formats for cloud computing interoperability. In: *International Conference on Internet and Web Applications and Services* (2009)
5. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) *ICA3PP 2010*. LNCS, vol. 6081, pp. 13–31. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13119-6_2
6. Cloud Standards (2017). <http://cloud-standards.org/>
7. ETSI Cloud Standards (2017). <http://www.etsi.org/newsevents/news/734-2013-12-press-release-report-on-cloudcomputing-standards>
8. Fehling, C., Mietzner, R.: Composite as a service: cloud application structures, provisioning, and management. *Inf. Technol.* **53**(4), 188–194 (2011)
9. Pahl, C., Jamshidi, P., Weyns, D.: Cloud architecture continuity: change models and change rules for sustainable cloud software architectures. *J. Softw. Evol. Process* **29**(2) (2017)
10. Pahl, C., Jamshidi, P., Zimmermann, O.: Architectural principles for cloud software. In: *ACM Transactions on Internet Technology*. (2018, to appear)
11. Fowley, F., Pahl, C., Zhang, L.: A comparison framework and review of service brokerage solutions for cloud architectures. In: *1st International Workshop on Cloud Service Brokerage* (2013)
12. Fowley, F., Pahl, C., Jamshidi, P., Fang, D., Liu, X.: A classification and comparison framework for cloud service brokerage architectures. *IEEE Trans. Cloud Comput.* (2017). <https://doi.org/10.1109/TCC.2016.2537333>. <http://ieeexplore.ieee.org/document/7423741/>
13. Garcia-Gomez, S., et al.: Challenges for the comprehensive management of cloud services in a PaaS framework. *Scalable Comput. Pract. Experience* **13**(3), 201–213 (2012)
14. Elango, D.M., Fowley, F., Pahl, C.: Pattern-driven architecting of an adaptable ontology-driven cloud storage broker. In: *University of Oslo, Department of Informatics, Research report 471*, pp. 33–47 (2017)
15. Gartner: Cloud Services Brokerage. Gartner Research (2013). <http://www.gartner.com/it-glossary/cloud-servicesbrokerage-csb>
16. Grozev, N., Buyya, R.: InterCloud architectures and application brokering: taxonomy and survey. *Softw. Pract. Experience* **44**(3), 369–390 (2012)
17. Pahl, C., Jamshidi, P.: Microservices: a systematic mapping study. In: *Proceedings CLOSER Conference*, pp. 137–146 (2016)
18. Taibi, D., Lenarduzzi, V., Pahl, C.: Processes, motivations and issues for migrating to microservices architectures: an empirical investigation. *IEEE Cloud Comput.* (2018). Accepted for publication

19. Hofer, C.N., Karagiannis, G.: Cloud computing services: taxonomy and comparison. *J. Internet Serv. Appl.* **2**(2), 81–94 (2011)
20. IEEE Cloud Standards (2015). <http://cloudcomputing.ieee.org/standards>
21. Jamshidi, P., Ahmad, A., Pahl, C.: Cloud migration research: a systematic review. *IEEE Trans. Cloud Comput.* **1**(2), 142–157 (2013)
22. jclouds: jclouds Java and Clojure Cloud API (2015). <http://www.jclouds.org/>
23. Ferrer, A.J., et al.: OPTIMIS: a holistic approach to cloud service provisioning. *Future Gener. Comput. Syst.* **28**(1), 66–77 (2012)
24. Gacitua-Decar, V., Pahl, C.: Structural process pattern matching based on graph morphism detection. *Int. J. Softw. Eng. Knowl. Eng.* **27**(2), 153–189 (2017)
25. Pahl, C.: Layered ontological modelling for web service-oriented model-driven architecture. In: Hartman, A., Kreische, D. (eds.) *ECMDA-FA 2005*. LNCS, vol. 3748, pp. 88–102. Springer, Heidelberg (2005). https://doi.org/10.1007/11581741_8
26. Pahl, C., Giesecke, S., Hasselbring, W.: Ontology-based modelling of architectural styles. *Inf. Softw. Technol.* **51**(12), 1739–1749 (2009)
27. Pahl, C., Xiong, H.: Migration to PaaS clouds - migration process and architectural concerns. In: *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA* (2013)
28. Konstantinou, A.V., Eilam, T., Kalantar, M., Totok, A.A., Arnold, W., Snibler, E.: An architecture for virtual solution composition and deployment in infrastructure clouds. In: *International Workshop on Virtualization Technologies in Distributed Computing* (2009)
29. Jamshidi, P., Sharifloo, A., Pahl, C., Arabnejad, H., Metzger, A., Estrada, G.: Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In: *12th International ACM SIGSOFT Conference on Quality of Software Architectures QoSA* (2016)
30. Arabnejad, H., Jamshidi, P., Estrada, G., El Ioini, N., Pahl, C.: An auto-scaling cloud controller using fuzzy Q-learning - implementation in openstack. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) *ESOCC 2016*. LNCS, vol. 9846, pp. 152–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_10
31. Mietzner, R., Leymann, F., Papazoglou, M.: Defining composite configurable SaaS application packages using SCA. In: *International Conference on Internet and Web Applications and Services, Variability Descriptors and Multi-tenancy Patterns* (2008)
32. Pahl, C., Xiong, H., Walshe, R.: A comparison of on-premise to cloud migration approaches. In: Lau, K.-K., Lamersdorf, W., Pimentel, E. (eds.) *ESOCC 2013*. LNCS, vol. 8135, pp. 212–226. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40651-5_18
33. Petcu, D., et al.: Portable cloud applications - from theory to practice. *Future Gen. Comput. Syst.* **29**(6), 1417–1430 (2013)
34. Javed, M., Abgaz, Y.M., Pahl, C.: Ontology change management and identification of change patterns. *J. Data Semant.* **2**(2–3), 119–143 (2013)
35. Amazon Simple Storage Service (S3) Cloud Storage AWS. <https://aws.amazon.com/s3/>
36. Dropbox. <https://www.dropbox.com/>
37. Azure Storage - Secure cloud storage. <https://azure.microsoft.com/en-us/services/storage/>
38. Google Drive - Cloud Storage & File Backup. <https://www.google.com/drive/>
39. Jamshidi, P., Pahl, C., Mendonca, N.C.: Pattern-based multi-cloud architecture migration. *Softw. Pract. Experience* **47**(9), 1159–1184 (2017)

40. Pahl, C., Brogi, A., Soldani, J., Jamshidi, P.: Cloud container technologies: a state-of-the-art review. *IEEE Trans. Cloud Comput.* (2017). <https://doi.org/10.1109/TCC.2017.2702586>. <http://ieeexplore.ieee.org/document/7922500/>
41. Aderaldo, C.M., Mendonca, N.C., Pahl, C., Jamshidi, P.: Benchmark requirements for microservices architecture research. In: 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering. *IEEE* (2017)
42. Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L.E., Pahl, C., Schulte, S., Wettinger, J.: Performance engineering for microservices: research challenges and directions. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion* (2017)



TosKER: Orchestrating Applications with TOSCA and Docker

Antonio Brogi, Luca Rinaldi, and Jacopo Soldani^(✉)

Department of Computer Science, University of Pisa, Pisa, Italy
soldani@di.unipi.it

Abstract. Docker is emerging as a simple yet effective solution for deploying and managing multi-component applications in virtualised cloud platforms. Application components can be shipped within portable and lightweight Docker containers, which can then be interconnected to allow components to interact each other. At the same time, the need for an enhanced support for orchestrating the management of the application components shipped within Docker containers is emerging.

In this paper we show how TOSCA can be exploited to provide such an enhanced support, by proposing a representation for describing the components forming an application, as well as the Docker containers used to ship such components. We also present TosKER, an engine for orchestrating the management of multi-component applications based on the proposed TOSCA representation and on Docker.

1 Introduction

Cloud computing has revolutionised IT, by allowing to run on-demand distributed applications at a fraction of the cost which was necessary just a few years ago [3]. This is possible as cloud providers exploit virtualisation techniques to achieve elasticity of large-scale shared resources [22]. Container-based virtualisation (where the operating system kernel permits running multiple isolated guest instances, called *containers*) can thus play an important role for cloud platforms, especially because it provides a lightweight virtualisation framework for PaaS/edge clouds [19,30]. Applications can be packaged, along with all software dependencies they need to run, into portable and lightweight containers, which can then be managed on cloud platforms [28].

Containers are also an ideal solution for SOA-based architectural patterns (e.g., microservices [24]) that are emerging in the cloud community to decompose monolithic applications into suites of independently deployable, lightweight components. Application components can indeed be packaged in independently deployable, lightweight containers, which can then be interconnected to allow components to interact with each other (forming multi-container applications [31]).

Docker [14] is considered the de-facto standard for container-based virtualisation [29]. Docker permits packaging software components in Docker *images*,

which are then exploited as read-only templates to create and run Docker *containers*. Docker containers can also mount external *volumes*, which ensure data persistence independently of the lifecycle of containers [23].

Docker permits orchestrating containers, by allowing to define multi-container Docker applications [31]. Given (the images of) the containers forming a multi-container application, the volumes they must mount, and the connections to set up among containers, Docker compose [15] is indeed capable of automatically deploying the corresponding application.

Docker containers are however treated as “black-boxes”, and they constitute the minimum orchestration entity considered by currently existing approaches for orchestrating multi-component applications with Docker (e.g., [2, 15, 16, 32]). Application components must be manually packaged, along with all their software dependencies, in (images of) Docker containers. Components are then strictly bound to their hosting containers, as it is not possible to orchestrate the management of the components forming an application independently of the Docker containers hosting them. For instance, it is not possible to run only some of the components hosted on a container, as whenever a container is started, all components it hosts are also started. Also, if we wish to change the container used to host a component, new Docker images must be manually developed (e.g., if a `maven` container is hosting the front-end and back-end of an application, and we wish to move the front-end to a `java` container, we must develop two new Docker images, one for hosting the front-end on a `java` container and one for hosting *only* the back-end on a `maven` container).

To fully exploit the potential of SOA, the current support for orchestrating multi-component applications with Docker should be enhanced. A concrete solution is to still rely on Docker containers as a portable and lightweight mean to deploy application components on cloud platforms, by also allowing to independently manage the components and containers forming a multi-component application [28]. In this paper we propose a solution precisely following this idea, which relies on the OASIS standard TOSCA [27] as the mean for orchestrating multi-component applications on top of Docker containers.

- We propose a TOSCA-based representation for multi-component applications, which permits modularly specifying the components forming an application, the Docker containers and Docker volumes needed to run them, as well as the relationships occurring among them (e.g., a component is hosted on a container, a component connects to another).
- We also present TOSKER, an engine for orchestrating the management of multi-component applications based on the proposed TOSCA representation and on Docker.

Our approach enhances the current support for orchestrating the management of multi-component applications in Docker, as it considers application components as orchestration entities, which are independent from the Docker containers and Docker volumes used to build their runtime infrastructure. For instance, TOSKER allows to independently manage the application components hosted on a Docker container, hence allowing to run only some of them

(if needed). Our approach also eases the change of Docker containers used to host the components of an application, as this only requires to update the corresponding TOSCA specification¹ (which will then be processed by TOSKER to automatically deploy and manage the specified application).

The rest of the paper is organised as follows. Section 2 provides some background on TOSCA. Section 3 illustrates our proposal for specifying multi-container Docker applications in TOSCA, and Sect. 4 presents the TOSKER engine for actually orchestrating such applications. Finally, Sects. 5 and 6 discuss related work and draw some concluding remarks, respectively.

2 Background

TOSCA (*Topology and Orchestration Specification for Cloud Applications* [27]) is an OASIS standard whose main goals are to enable (i) the specification of portable cloud applications and (ii) the automation of their deployment and management. TOSCA provides a YAML-based and machine-readable modelling language that permits describing cloud applications. Obtained specifications can then be processed to automate the deployment and management of the specified applications. We hereby report only those features of the TOSCA modelling language that are used in this paper².

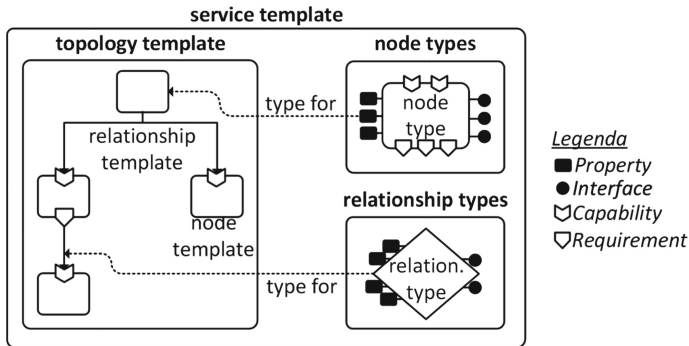


Fig. 1. The TOSCA metamodel [27].

TOSCA permits specifying a cloud application as a service template, that is in turn composed by a topology template, and by the types needed to build such a topology template (Fig. 1). The topology template is a typed directed graph that

¹ This can also be done automatically by exploiting TOSKERISER [10]. Given a TOSCA application specification, TOSKERISER can indeed automatically (discover and) include the Docker containers offering the software support needed by its components.

² A more detailed, self-contained introduction to TOSCA can be found in [5, 12].

describes the topological structure of a multi-component application. Its nodes (called node templates) model the application components, while its edges (called relationship templates) model the relations occurring among such components.

Node templates and relationship templates are typed by means of node types and relationship types, respectively. A node type defines the observable properties of a component, its possible requirements, the capabilities it may offer to satisfy other components' requirements, and the interfaces through which it offers its management operations. Requirements and capabilities are also typed, to permit specifying the properties characterising them. A relationship type instead describes the observable properties of a relationship occurring between two application components. As the TOSCA type system supports inheritance, a node/relationship type can be defined by extending another, thus permitting the former to inherit the latter's properties, requirements, capabilities, interfaces, and operations (if any).

Node templates and relationship templates also specify the artifacts needed to actually realise their deployment or to implement their management operations. As TOSCA allows artifacts to represent contents of any type (e.g., scripts, executables, images, configuration files, etc.), the metadata needed to properly access and process them is described by means of artifact types.

TOSCA applications are then packaged and distributed in CSARs (*Cloud Service ARchives*). A CSAR is essentially a zip archive containing an application specification along with the concrete artifacts realising the deployment and management operations of its components.

3 Specifying Multi-component Applications

Multi-component applications typically integrate various and heterogeneous components [18]. We hereby define a TOSCA-based representation for such components, as well as for the Docker containers and Docker volumes that will be used to form their runtime infrastructure.

We first define three different TOSCA node types³ to permit distinguishing the Docker containers, the Docker volumes, and the application components forming a multi-component application (Fig. 2).

- *tosker.nodes.Container* permits representing Docker containers, by indicating whether a container requires a *connection* (to another Docker container or to an application component), whether it has a generic *dependency* on another node in the topology, or whether it needs some persistent *storage* (hence requiring to be attached to a Docker volume). *tosker.nodes.Container* also permits indicating whether a container can *host* an application component, whether it offers an *endpoint* where to connect to, or whether it offers a generic *feature* (to satisfy a generic *dependency* requirement of another

³ The actual definition of all TOSCA types discussed in this section is publicly available on GitHub at <https://github.com/di-unipi-socc/tosker-types>.

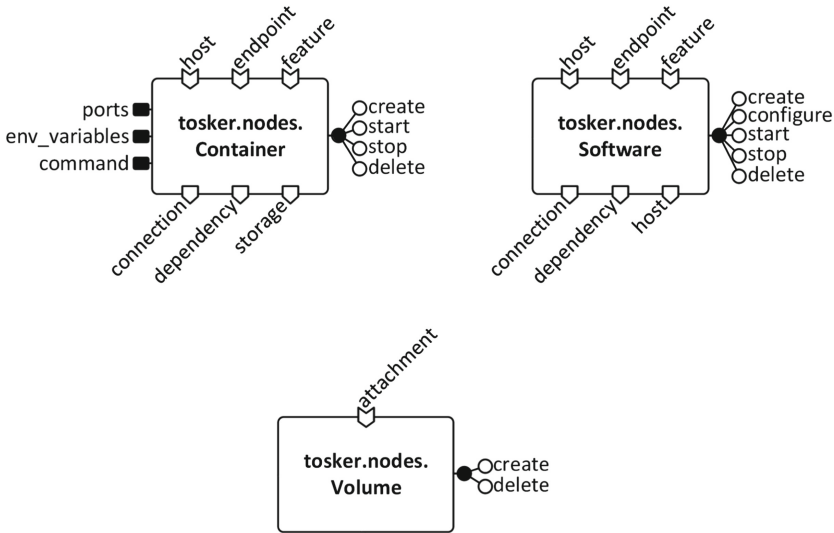


Fig. 2. TOSCA node types for multi-component, Docker-based applications, viz., *tosker.nodes.Container*, *tosker.nodes.Software*, and *tosker.nodes.Volume*.

container/application component). To complete the description, *tosker.nodes.Container* provides placeholders (through the properties *ports*, *env_variables* and *command*, respectively) for specifying the port mappings, the environment variables, and the command to be executed when running the corresponding Docker container, and it lists the operations to manage a container (which correspond to the basic operations offered by the Docker platform [23]).

- *tosker.nodes.Volume* permits specifying Docker volumes, and it defines a capability *attachment* to indicate that a Docker volume can satisfy the *storage* requirements of Docker containers. It also lists the operations to manage a Docker volume (which corresponds to the basic operations offered by the Docker platform [23]).
- *tosker.nodes.Software* permits indicating the software components forming a multi-component application. It permits specifying whether an application component requires a *connection* (to a Docker container or to another application component), whether it has a generic *dependency* on another node in the topology, and that it has to be *hosted* on a Docker container or on another component⁴. *tosker.nodes.Software* also permits indicating whether an application component can *host* another application component, whether it provides an *endpoint* where to connect to, or whether it offers a generic *feature* (to satisfy a generic *dependency* requirement of a container/application

⁴ The *host* requirement is mandatory for nodes of type *tosker.nodes.Software*, as we assume that each application component must be installed in another component or in a Docker container.

component). Finally, *tosker.nodes.Software* indicates the operations to manage an application component (viz., *create*, *configure*, *start*, *stop*, *delete*).

The interconnections and interdependencies among the nodes forming a multi-component application can be indicated by exploiting the TOSCA normative relationship types [27].

- *tosca.relationships.AttachesTo* can indeed be used to attach a Docker volume to a Docker container.
- *tosca.relationships.ConnectsTo* can indicate the network connections to establish between Docker containers and/or application components.
- *tosca.relationships.HostedOn* can be used to indicate that an application component is hosted on another component or on a Docker container (e.g., to indicate that a web service is hosted on a web server, which is in turn hosted on a Docker container).
- *tosca.relationships.DependsOn* can be used to indicate generic dependencies between the nodes of a multi-component application (e.g., to indicate that a component must be deployed before another, as the latter depends on the availability of the former to properly work).

Example 1. Consider *Thinking*, an open-source⁵ web application that allows users to share their thoughts, so that all other users can read them. *Thinking* is composed by three main components, namely (i) a Mongo database storing the collection of thoughts shared by end-users, (ii) a Java-based REST API to remotely access the database of shared thoughts, and (iii) a web-based GUI visualising all shared thoughts and allowing to insert new thoughts into the database. Figure 3 illustrates a representation of the *Thinking* application in TOSCA.

- (i) The database is obtained by directly instantiating a *MongoDB* container, which needs to be attached to a volume where the shared thoughts will be persistently stored.
- (ii) The *API* is hosted on a *Maven* Docker container, and it requires to be connected to the *MongoDB* container (for remotely accessing the database of shared thoughts).
- (iii) The *GUI* is hosted on a *NodeJS* Docker container, and it depends on the availability of the *API* to properly work (as it sends GET/POST requests to the *API* to retrieve/add shared thoughts). □

Finally, also artifacts must be typed [27], as they are used to implement deployment and management operations of the nodes forming a multi-component application and they must specify the metadata needed to properly access and process them. We hence define *tosker.artifacts.Image* and *tosker.artifacts.Dockerfile* to permit indicating that an artifact is an actual image or a Dockerfile, which will then be used to create a Docker container. We also extend such artifact types by defining *tosker.artifacts.Image.Service* and *tosker.artifacts.Dockerfile.Service*,

⁵ The source code of *Thinking* is publicly available on GitHub at <https://github.com/di-unipi-socc/thinking>.

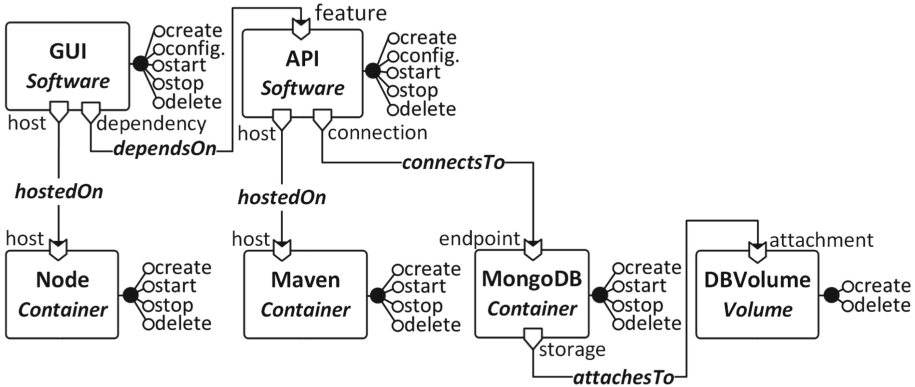


Fig. 3. An example of multi-component application specified in TOSCA (where nodes are typed with *tosker.nodes.Container*, *tosker.nodes.Volume*, or *tosker.nodes.Software*, while relationships are typed with TOSCA normative types [27]).

to permit distinguishing images that execute a service when started from those that “simply package” a runtime environment. We can instead rely on TOSCA normative artifact types [27] for all other kinds of artifacts linked by the nodes in a multi-container Docker application.

Example 1 (cont.). Consider again the application in Fig. 3. The image artifact associated to the *MongoDB* container is of type *tosker.artifacts.Image.Service*, as it links to an image offering a MongoDB server when executed. The image artifacts associated to the containers *Node* and *Maven* are instead of type *tosker.artifacts.Image*, as they link to images just offering runtime environments (for NodeJS-based and Maven-based applications, respectively). The management operations of *GUI* and *API* are instead implemented by “.sh” scripts⁶. □

4 TOSKER

We hereby present TOSKER, an orchestrator capable of automatically deploying and managing multi-component applications specified with the proposed TOSCA representation. We first illustrate the architecture of TOSKER, and we then discuss its current prototype implementation.

⁶ The resulting TOSCA application specification is publicly available at <https://github.com/di-unipi-socc/TosKer/blob/master/data/examples/thoughts-app/thoughts/thoughts.yaml>. A CSAR packaging such specification (together with all artifacts needed to deploy and manage the *Thinking* application) is available at <https://github.com/di-unipi-socc/TosKer/blob/master/data/examples/thoughts-app/thoughts.csar>.

4.1 The Architecture of TOSKER

Figure 4 shows the architecture of TOSKER, which is designed to be modular and easily extensible. The architecture of TOSKER indeed partitions the functionalities of TOSKER into lightweight modules that interact with each other, and new functionalities can be easily added to TOSKER by developing and plugging-in new modules.

User interface. The UI allows to feed TOSKER with the necessary input. The latter includes a CSAR (packaging the TOSCA specification of a multi-component application together with all artifacts needed to realise its management), a sequence of management operations to be executed, and (optionally) the subset of the application components on which to perform such a sequence of management operations.

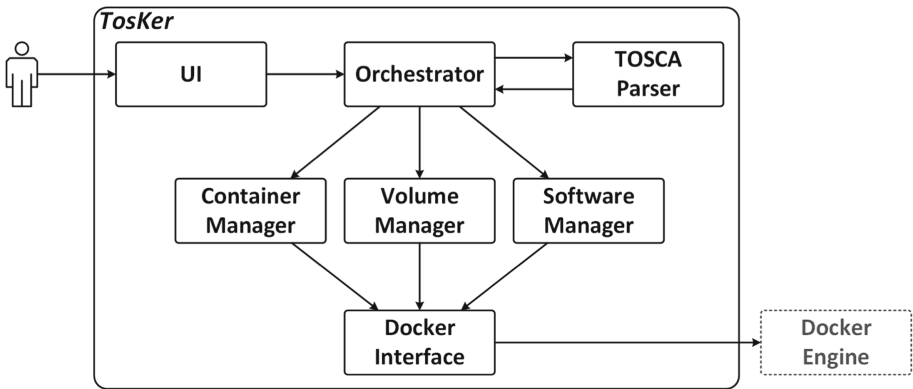


Fig. 4. The architecture of TOSKER.

TOSCA utilities. The TOSCA Parser is an utility module for parsing a CSAR and generating an internal representation of the application it packages. Such representation will then be exploited by the other modules in TOSKER to deploy and manage the corresponding application.

Orchestration core. The Orchestrator is the core component of TOSKER, as it is in charge of planning and orchestrating the management of multi-component applications. It first receives the input from the UI, and it exploits the TOSCA Parser to generate an internal representation of the multi-component application contained in the input CSAR.

The Orchestrator automatically determines which management operations have to be executed on which components, and in which order⁷. (to permit

⁷ The Orchestrator assumes that components are managed according to the TOSCA standard management lifecycle [27]. If such lifecycle is not respected (e.g., by requiring to *delete* a component that has not yet been created), then the Orchestrator will raise an error and stop orchestrating the application management.

executing the input sequence of operations on the indicated subset of application components). The result is a (possibly expanded) sequence of management operations, each to be executed on a certain application component.

The **Orchestrator** then orchestrates the actual execution the above mentioned sequence of management operations by coordinating the **Container Manager**, **Volume Manager** and **Software Manager**. It indeed iterates over the sequence, and it dispatches the actual execution of an operation on a component to the corresponding manager (e.g., to *create* a component of type *tosker.nodes.Container*, the Orchestrator dispatches the actual execution of *create* on such component to the Container Manager). dispatched to the

Managers. The Container Manager, Volume Manager, and Software Manager implement the actual lifecycle for components of type *tosker.nodes.Container*, *tosker.nodes.Volume*, and *tosker.nodes.Software*, respectively.

- The **Container Manager** is in charge of implementing the operations to *create*, *start*, *stop* and *delete* Docker containers, by also taking into account the different types of artifacts from which they are generated (viz., Docker images or Dockerfiles—see Sect. 3).
- The **Volume Manager** has to implement the operations to *create* and *delete* Docker volumes (as volumes can only be created or deleted [23]).
- The **Software Manager** is in charge of implementing the operations to *create*, *configure*, *start*, *stop* and *delete* a component of type *tosker.nodes.Software*. Notice that, as such a kind of components will be hosted on Docker containers, the actual execution of a management operation on a component requires to issue commands to its container. For instance, to *create* a component, the Software Manager has to (i) copy all artifacts of the component inside a dedicated folder of its container, (ii) start the container by executing the script implementing the *create* operation of the component, (iii) commit the changes applied to the container as a new image, and (iv) re-create the container by exploiting the newly created image.

Notice that each manager implements management operations by instructing the Docker Interface on which Docker commands to execute.

Docker interface. The Docker Interface is in charge of interacting with the Docker engine installed on the host where TOSKER is running. It is used by the managers to manage Docker containers and Docker volumes, and to execute operations inside running containers.

Notice that the Docker Interface decouples TOSKER from the actual Docker engine used, meaning that it can issue commands to a classic Docker engine (as in the current implementation of TOSKER—see Sect. 4.2), but it could also be used to issue commands to an engine capable of distributing containers in a cluster (e.g., Docker swarm [16] or Kubernetes [32]).

4.2 Prototype Implementation

We have implemented a prototype of TOSKER, which is open-source and publicly available on GitHub⁸. The prototype is written in Python⁹, and it is composed by a main package (`tosker`) containing the set of Python modules implementing the various components forming the architecture of TOSKER (viz., `ui.py`, `tosca_parser.py`, `orchestrator.py`, `container_manager.py`, `volume_manager.py`, `software_manager.py`, and `docker_interface.py`).

The current prototype of TOSKER is also published on PyPI¹⁰ (*Python Package index*), which permits installing it on a host by simply executing the command `pip install tosker`. It can then be used as a standard Python library, or as a command line software by executing:

```
$ tosker FILE [COMPONENTS] COMMANDS [INPUTS]
```

where `FILE` is a CSAR archive or a TOSCA YAML file (containing the specification of a multi-component application), `COMPONENTS` is optional and permits specifying the subset of application components to be managed, `COMMANDS` is the sequence of management operations to be executed, and `INPUTS` is an optional sequence of input parameters to be passed to the TOSCA application¹¹.

Example 2. Consider again the *Thinking* application in Example 1. Suppose, for instance, that we wish to *create* and *start* its *API* and *MongoDB*. We can instruct TOSKER to do so, by executing:

```
$ tosker /usr/share/tosker/examples/thoughts.csar \  
API MongoDB create start
```

Notice that this will not only result in creating and starting *API* and *MongoDB*, but also the *Maven* container and *DBVolume* they require to properly work. *GUI* and *Node* will instead be ignored by TOSKER, as they are not contained in set of components input to TOSKER, nor they are needed by *API* or *MongoDB*. □

To test the current prototype of TOSKER, we specified the open-source application *Thinking* in TOSCA, as well as three other existing applications, viz., (i) a Wordpress instance running on a PHP web server and connecting to a MySQL back-end, (ii) a NodeJS-based REST API connecting to a MongoDB back-end, and (iii) an application with three interacting servers written in NodeJS. All applications were effectively deployed by the current prototype of TOSKER, and they constituted the basis for developing a battery of unit tests¹², which covered 96% of the source code of the Python modules we implemented (see Table 1).

⁸ <https://github.com/di-unipi-socc/TosKer>.

⁹ The choice of Python was mainly motivated by the availability of two open-source Python libraries: *docker-py* (<https://github.com/docker/docker-py>) and *tosca-parser* (<https://github.com/openstack/tosca-parser/>). *docker-py* implements a

Table 1. Unit test coverage in the current prototype of TOSKER (obtained by running the *coverage-py* tool—<https://coverage.readthedocs.io>).

| Module | Total statements | Missed statements | Coverage |
|-----------------------------------|------------------|-------------------|----------|
| <code>ui.py</code> | 75 | 18 | 76% |
| <code>docker_interface.py</code> | 168 | 4 | 98% |
| <code>tosca_parser.py</code> | 219 | 2 | 99% |
| <code>orchestrator.py</code> | 105 | 2 | 98% |
| <code>container_manager.py</code> | 26 | 0 | 100% |
| <code>volume_manager.py</code> | 9 | 0 | 100% |
| <code>software_manager.py</code> | 67 | 2 | 97% |
| Total | 669 | 28 | 96% |

5 Related Work

We hereby position TOSKER with respect to other currently available solutions for orchestrating the management of multi-component applications with Docker and/or TOSCA.

Docker-based orchestration. Docker natively supports multi-container Docker applications with Docker compose [15]. Docker compose permits specifying the (images of) containers forming an application, the links/connections to be set between such containers, and the volumes to be mounted. Based on that, Docker compose is capable of deploying the specified application. However, Docker compose treats containers as black-boxes, meaning that there is no information on which components are hosted by a container, and that it is not possible to orchestrate the management of application components separately from that of their containers (as it is instead possible with TOSKER).

Other approaches worth mentioning are Docker swarm [16], Kubernetes [32], and Mesos [2]. Docker swarm permits creating a cluster of replicas of a Docker container, and seamlessly managing it on a cluster of hosts. Kubernetes and Mesos instead permit automating the deployment, scaling, and management of containerised applications over clusters of hosts. Docker swarm, Kubernetes and Mesos differ from TOSKER as they focus on how to schedule and manage containers on clusters of hosts, rather than on how to orchestrate the management of the components and containers forming multi-component applications.

Python interface for the Docker engine API. *tosca-parser* is instead a parser for TOSCA application specifications (developed by the OpenStack community).

¹⁰ <https://pypi.python.org/pypi/tosker>.

¹¹ Details on how to process inputs for TOSCA applications can be found in [27].

¹² The TOSCA application specifications and the battery of unit tests that we implemented are publicly available on GitHub at <https://github.com/di-unipi-socc/Tosker/tree/master/data/examples> and <https://github.com/di-unipi-socc/Tosker/tree/master/tests>, respectively.

TOSCA-based orchestration. OpenTOSCA [4] is an open-source engine for deploying and managing TOSCA applications. It is designed to work with a former, XML-based version of TOSCA [25], and to process applications “imperatively” (viz., by executing management plans defined by the application developer in the form of BPEL or BPMN workflows). TOSKER instead works with the newer, YAML-based version of TOSCA [27], and it is designed to process applications “declaratively” (viz., by automatically determining the management plans to be executed from the topology of an application).

Other approaches worth mentioning are SeaClouds [8], Brooklyn [1], Alien4-Cloud [17], and Cloudify [20]. SeaClouds [8] is a middleware solution for deploying and managing multi-component applications on heterogeneous IaaS/PaaS clouds. SeaClouds fully supports TOSCA, but it lacks a support for Docker containers. The latter makes SeaClouds not suitable to orchestrate the management of multi-component applications including Docker containers.

Brooklyn [1], Alien4Cloud [17] and Cloudify [20] instead natively support Docker containers, and they permit orchestrating the management of the software components and Docker containers forming cloud applications. They however all differ from TOSKER because they treat Docker containers as black-boxes (hence not permitting to orchestrate the management of application components separately from that of the containers hosting them).

Brooklyn [1] and Cloudify [20] also differ from TOSKER as they require to specify applications in non-standard blueprint languages (inspired to, but not fully compliant with, the OASIS standards CAMP [26] and TOSCA [26], respectively). For instance, a relationship is specified in TOSCA by connecting a requirement of one component to a capability of another, and requirements/capabilities can be used to express interconnection constraints (which then permit validating TOSCA application topologies [9]). Cloudify blueprints instead do not include any notion of requirements or capabilities, as relationships just connect a source node to a target node.

Summary. To the best of our knowledge, ours is the first solution that permits specifying and orchestrating multi-component, Docker-based applications in TOSCA, and managing software components independently of the containers hosting them.

6 Conclusions

Container-based virtualisation is emerging as a simple yet effective solution for deploying and managing multi-component applications in cloud platforms [28]. Application components can be shipped within portable and lightweight Docker containers, which can then be interconnected to allow components to interact with each other. At the same time, the current support for orchestrating the management of the application components shipped within Docker containers is limited [29]. For instance, components must be manually packaged in Docker containers, and it is not possible to manage components independently of the

containers hosting them (e.g., whenever a container is started/stopped, all components hosted on such container are also started/stopped).

In this paper we illustrated how TOSCA [27] can enhance the support for orchestrating multi-component applications with Docker. We indeed (i) proposed a TOSCA-based representation for multi-component applications, which permits distinguishing the Docker containers and software components in a multi-component application, as well as the relationships occurring among them. We also (ii) presented TOSKER, an orchestration engine for automatically deploying and managing multi-component applications based on TOSCA and Docker.

Our approach enhances the current support for orchestrating the management of multi-component applications in Docker. TOSKER can indeed automatically install application components within the containers hosting them (instead of requiring to manually package components in images of Docker containers), and it permits independently orchestrating the management of components and containers (instead of binding the management lifecycle of components to that of the containers hosting them).

We believe that our approach can also facilitate the widespread adoption of the TOSCA standard. TOSKER indeed provides a lightweight, easy-to-use engine for deploying and managing TOSCA-based applications (exploiting Docker to host their components).

We tested the current prototype of TOSKER by developing a battery of unit tests based on four existing applications. A more thorough evaluation of TOSKER, based on concrete case studies and/or on datasets of multi-component applications (e.g., μ SET [6]), is in the scope of our immediate future work.

Additionally, the current prototype of TOSKER permits orchestrating applications on single hosts and it does not yet support horizontal scaling of containers. TOSKER can be adapted to include such features, for instance, by simply including a new version of the Docker Interface which interacts with Docker Swarm [16] or Kubernetes [32] (instead of with the Docker engine installed on a host). This is also in the scope of our future work.

It is finally worth noting that TOSKER permits orchestrating the management of multi-component applications, by already offering some basic planning capabilities. For instance, when required to *start* a component of an application, TOSKER automatically determines which other components have to be started, and it plans the sequence of operations that permits starting all such components. Such planning is however based on a fixed set of operations, whose behaviour is fixed by the TOSCA standard management lifecycle [27]. This is because our approach does not yet include a way to customise the management behaviour of application components. A solution can be to integrate our approach with models designed precisely to permit compositionally describing the management behaviour of the components forming an application (e.g., Aeolus [13] or fault-aware management protocols [7]), which would also permit improving the planning capabilities of TOSKER (e.g. by exploiting the Aeolus-based planning algorithm in [21]). The integration of our approach with an existing solution for modelling, analysing and planning the management of multi-component applications is also in the scope of our future work.

Acknowledgments. The authors would like to thank Claus Pahl for all helpful and stimulating discussions on how to enhance the current support for orchestrating multi-component applications with Docker, which were reported in [11] and laid the foundations for the work presented in this paper.

References

1. Apache Software Foundation: Brooklyn. <http://brooklyn.apache.org>
2. Apache Software Foundation: Mesos. <http://mesos.apache.org/>
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. *Commun. ACM* **53**(4), 50–58 (2010)
4. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) *ICSOC 2013*. LNCS, vol. 8274, pp. 692–695. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_62
5. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) *Advanced Web Services*, pp. 527–549. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7535-4_22
6. Brogi, A., Canciani, A., Neri, D., Rinaldi, L., Soldani, J.: Towards a reference dataset of microservice-based applications. In: Cerone, A., Roveri, M. (eds.) *SEFM 2017*. LNCS, vol. 10729, pp. 219–229. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74781-1_16
7. Brogi, A., Canciani, A., Soldani, J.: Fault-aware application management protocols. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) *ESOCC 2016*. LNCS, vol. 9846, pp. 219–234. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_14
8. Brogi, A., Carrasco, J., Cubo, J., D’Andria, F., Ibrahim, A., Pimentel, E., Soldani, J.: EU Project SeaClouds - adaptive management of service-based applications across multiple clouds. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*, pp. 758–763 (2014)
9. Brogi, A., Di Tommaso, A., Soldani, J.: Validating TOSCA application topologies. In: *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, MODELWARD, vol. 1*, pp. 667–678. SciTePress (2017)
10. Brogi, A., Neri, D., Rinaldi, L., Soldani, J.: From (incomplete) TOSCA specifications to running applications, with Docker. In: Cerone, A., Roveri, M. (eds.) *SEFM 2017*. LNCS, vol. 10729, pp. 491–506. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74781-1_33
11. Brogi, A., Pahl, C., Soldani, J.: Enhancing the orchestration of multi-container Docker applications (2016). Submitted for Publication
12. Brogi, A., Soldani, J., Wang, P.W.: TOSCA in a Nutshell: promises and perspectives. In: Villari, M., Zimmermann, W., Lau, K.-K. (eds.) *ESOCC 2014*. LNCS, vol. 8745, pp. 171–186. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44879-3_13
13. Di Cosmo, R., Mauro, J., Zacchiroli, S., Zavattaro, G.: Aeolus: a component model for the cloud. *Inf. Comput.* **239**, 100–121 (2014)

14. Docker Inc.: Docker. <https://www.docker.com/>
15. Docker Inc.: Docker compose. <https://github.com/docker/compose>
16. Docker Inc.: Docker swarm. <https://github.com/docker/swarm>
17. FastConnect, Bull, Atos: Alien4cloud. <https://alien4cloud.github.io/>
18. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, Vienna (2014). <https://doi.org/10.1007/978-3-7091-1568-8>
19. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171–172. IEEE Computer Society (2015)
20. GigaSpaces Technologies: Cloudify. <http://cloudify.co/>
21. Lascu, T.A., Mauro, J., Zavattaro, G.: A planning tool supporting the deployment of cloud applications. In: Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, ICTAI 2013, pp. 213–220. IEEE Computer Society (2013)
22. Leymann, F.: Cloud computing. it – Information Technology, Methoden und innovative Anwendungen der Informatik und Informationstechnik **53**(4), 163–164 (2011)
23. Matthias, K., Kane, S.P.: Docker: Up and Running. O’Reilly Media, Sebastopol (2015)
24. Newman, S.: Building Microservices. O’Reilly Media, Inc., Sebastopol (2015)
25. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA), Version 1.0 (2013). <http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf>
26. OASIS: Cloud Application Management for Platforms (CAMP), Version 1.1 (2016). <http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf>
27. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Simple Profile in YAML, Version 1.0 (2016). <http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf>
28. Pahl, C.: Containerization and the paas cloud. IEEE Cloud Comput. **2**(3), 24–31 (2015)
29. Pahl, C., Brogi, A., Soldani, J., Jamshidi, P.: Cloud container technologies: a state-of-the-art review. IEEE Trans. Cloud Comput. (in press). <https://doi.org/10.1109/TCC.2017.2702586>. Early access: <http://ieeexplore.ieee.org/document/7922500/>
30. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures - a technology review. In: Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud, FICLOUD 2015, pp. 379–386. IEEE Computer Society (2015)
31. Smith, R.: Docker Orchestration. Packt Publishing, Birmingham (2017)
32. The Kubernetes Authors: Kubernetes. <http://kubernetes.io/>

EU Projects

Preface of EU Projects Track 2017

The third edition of the EU Projects Track held at ESOCC 2017 was entirely devoted to presenting results and perspectives of EU research projects on service-oriented and cloud computing.

As in the previous two editions, the track provided a very good opportunity to presenters for disseminating the results of their project, and to participants for getting an updated view of the ongoing research on service-oriented and cloud computing.

These proceedings contain the descriptions of eight EU projects (BASMATI, C4E, DICE, DITAS, HyVar, MIKELANGELO, RestAssured, SWITCH) that were presented in Oslo. Each project description was (anonymously) reviewed by three members of the Program Committee formed for the track.

I would like to thank authors, Program Committee members, and attendees of the track, and the local Organizing Committee of ESOCC 2017 for the strong support.

Antonio Brogi

Organization

Track Chair

Antonio Brogi University of Pisa, Italy

Program Committee

| | |
|---------------------|---|
| Marco Aiello | University of Groningen, The Netherlands |
| Benoit Baudry | Inria, France |
| Einar Broch Johnsen | University of Oslo, Norway |
| Giuliano Casale | Imperial College, UK |
| Paul Grefen | Eindhoven University of Technology, The Netherlands |
| Philippe Massonet | CETIC, Belgium |
| Ernesto Pimentel | University of Malaga, Spain |
| Lutz Schubert | Ulm University, Germany |
| Jacopo Soldani | University of Pisa, Italy |
| Massimo Villari | University of Messina, Italy |
| Gianluigi Zavattaro | University of Bologna, Italy |



Secure Data Processing in the Cloud

Zoltán Ádám Mann¹, Eliot Salant², Mike SurrIDGE³(✉),
Dhouha Ayed⁴, John Boyle⁵, Maritta Heisel¹, Andreas Metzger¹,
and Paul Mundt⁶

¹ University of Duisburg-Essen, Duisburg, Germany

² IBM Haifa Research Labs, Haifa, Israel

³ University of Southampton IT Innovation Centre, Southampton, UK
ms@it-innovation.soton.ac.uk

⁴ Thales Services, Palaiseau, France

⁵ Oxford Computer Consultants, Oxford, UK

⁶ Adaptant Solutions AG, Munich, Germany

<http://www.restassuredh2020.eu/>

Abstract. Data protection is a key issue in the adoption of cloud services. The project “RestAssured – Secure Data Processing in the Cloud,” financed by the European Union’s Horizon 2020 research and innovation programme, addresses the challenge of data protection in the cloud with a combination of innovative security solutions, data lifecycle management techniques, run-time adaptation, and automated risk management. This paper gives an overview about the project’s goals and current status.

Keywords: Cloud computing · Data protection · Privacy
Secure hardware enclaves · Sticky policies · Run-time adaptation
Automated risk management

1 Project Objectives

Secure cloud computing is key for business success and end user adoption of federated and decentralized cloud services, and as such, is essential to stimulating the growth of the European Digital Single Market. And, while cloud-based data be kept secure, these data must also be made accessible to authorized users while ensuring privacy regulations such as the European Union’s General Data Protection Regulation (GDPR)¹.

The RestAssured project aims to provide solutions to specific technical concerns of data protection in the cloud through four main areas of innovation (see Fig. 1):

- Use of emerging hardware solutions such as Intel’s SGX to provide secure enclaves for data operations.
- Implementation of sticky policies which define data access, usage and storage rules.
- Run-time data protection assurance using self-adaptation and models@runtime.
- Automated risk management to automatically detect risks to data protection and rapidly determine the cost vs. benefits of alternative protection mechanisms.

¹ <http://www.eugdpr.org/>.

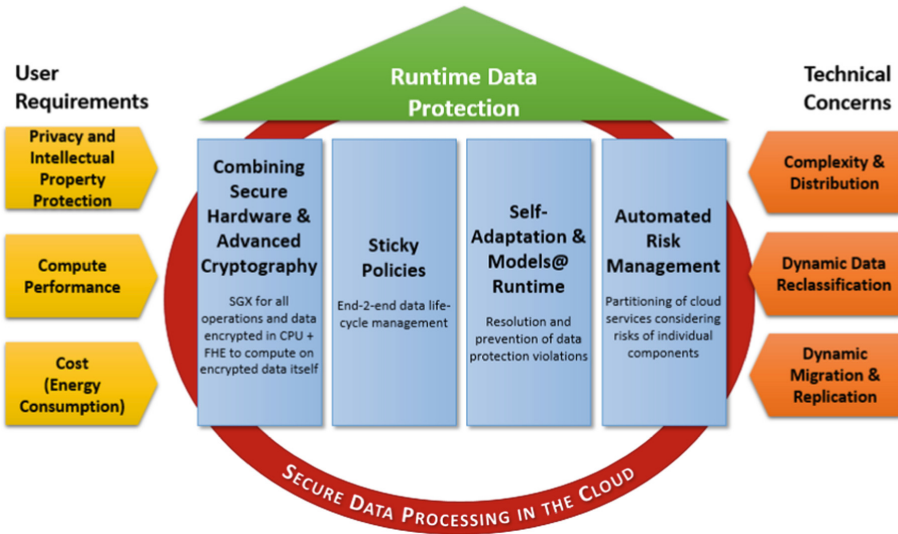


Fig. 1. The RestAssured pillars of innovation

1.1 Secure Enclaves

Secure enclaves are offered by Intel’s SGX (Software Guard Extensions) which is currently available in the marketplace, or AMD’s SME (Secure Memory Encryption). While SGX and SME use different approaches, each with its advantages and disadvantages, the general idea is the same: a memory range is encrypted by the processor by a key which is generated at power-on, and not available to any running process. This means that all code and data within an enclave are protected from tamper and snooping, even by processes running at superuser level, or by dumping memory.

RestAssured is creating a toolkit which will significantly simplify the work required by a developer to set up and use an SGX enclave (such as remote attestation, sealing, secret passing), allowing developers to focus on the development of their business logic. Additionally, RestAssured is integrating the Open Source Opaque project into its SGX toolkit. Opaque is a Spark SQL engine that can work with encrypted data, leveraging Intel SGX to protect the computation [1]. Users can run SQL queries in a Spark shell, or program the queries in high-level Scala language. There is no need to develop SGX applications in C/C++ with the SGX SDK. However, Opaque has some design and implementation limitations, related to attestation and data key passing. By integrating with the RestAssured toolkit, we enable efficient attestation of Opaque enclaves, flexible data key passing and overall integration into the RestAssured platform.

1.2 Sticky Policies

Sticky policies for data define access rights on the data and, as their name suggests, “stick” to the data, following it as it migrates across the cloud. In this way, sticky

policies allow for decentralized data lifecycle management; i.e. access control can be enforced by decision points across the cloud, without the need for a centralized enforcement entity. Sticky policies need not only support the rights of the data subject in accordance with GDPR requirements, but must also be able to support the security and privacy regulations which may be mandated by the enterprise which either owns or processes the data, as well as any regulations the data may be subject to, based on its physical storage location across the cloud.

1.3 Run-Time Data Protection Assurance

The flexibility and dynamism of the cloud poses a big challenge for data protection. The applicability of traditional security mechanisms designed to keep the system in a stable secure state is limited. In particular, security-by-design methodologies are not sufficient, due to uncertainty at design time as to how the cloud and privacy requirements may evolve and change at run time.

To cope with continuously changing data protection requirements in a continuously changing cloud environment, we apply methods from the field of self-adaptive systems [2, 3]. This way the system can adapt to changes in both the cloud and the data protection requirements, ensuring that requirements are met in the presence of changes, with minimal impact on performance and costs. Adaptations may be made either fully automatically, or after approval from a human operator.

To make sound adaptation decisions at run time, one needs a model of the system, its requirements and environment. The model must be available at run time to enable online reasoning, hence it is called *model@runtime* [4]. Data protection concerns relate to all layers of the cloud stack, including secure hardware capabilities, co-location of different tenants on the same server, encryption of communication between application components, and data anonymization. All must be captured by the *model@runtime*. By monitoring the state of the system and its environment, updating the model, and comparing observed behavior to expected behavior, violations of data protection policies can be detected. If a violation is detected, further reasoning using the *model@runtime* can be used to automatically find and execute an appropriate adaptation action. This way, data protection issues can be mitigated or prevented automatically.

1.4 Automated Risk Management

Under the GDPR, personal data controllers and processors must assess risks to personal data, and employ security measures to appropriately manage identified risks, throughout the life of the system(s) storing and using the data. Moreover, the data controller is responsible for proving the systems and processes used comply with the GDPR. It is no longer enough to implement recommended security measures based on a ‘generic’ risk analysis – one must analyze risks specific to each situation, design systems and processes to address risks to privacy, and continuously review and update the risk analysis and security measures as either the system or the threat landscape evolves.

The requirement for continuous and auditable management of risks is especially difficult in cloud-based applications, which may be subject to automatic adaptation at any time. One of the main goals of RestAssured is to provide the means to trace how such changes affect the level of risk, and where changes are made specifically to manage risks, e.g. by allocating sensitive processes to a secure enclave, or by introducing advanced encryption to block new risks when migrating workloads. The goal is to provide technologies that help data controllers to analyze risks and trace the measures used to address risks, making it much easier to comply with GDPR when using cloud-based applications. To achieve this, RestAssured integrates and extends two innovative approaches to information risk analysis:

- The Cloud System Analysis Pattern methodology [5] developed by University of Duisburg-Essen in the CloudDAT project to help stakeholders identify socio-technical assets and carry out a risk analysis specifically focusing on cloud applications;
- A procedure devised by IT Innovation [6, 7] in the SERSCIS, OPTET and UK ASSURED projects that automates risk identification and analysis based on a description of a system in terms of its assets, and supports the selection of risk management measures.

The use of an asset-based risk analysis approach supports compliance with information risk management standards like ISO 27001 [8], while the use of automation based on machine reasoning makes it possible to perform this analysis on a continuous basis in the loop of autonomic cloud application and infrastructure management processes.

2 Project Current State and Summary of Results

Although the project is still in its first year, good progress has been made towards having a prototype implementation running two real-world use cases by project month 18.

The first use case, highlighting social care services, shows how volunteer healthcare workers can be matched with suitable healthcare patients in a secure environment, preserving the data access rights specified by both parties. Additionally, this use case demonstrates how RestAssured can enforce the security and privacy requirements for a workflow specifying the generation of summary reports by a third party only allowing them access to anonymized data. Pivotal to this use case is the ability to integrate Opaque into the RestAssured environment to support the database queries required to match caregivers with patients, as well as sticky policy enforcement across the whole workflow of the data.

An additional use case demonstrates a “pay-as-you-drive” scenario – where a driver’s insurance rates depend on their monitored driving behavior. In this scenario, an application at the network edge (e.g. Connected Car) enables the data subject (driver) to identify and limit the transfer of personally identifiable information to the service provider for providing an agreed upon service (e.g. usage-based car insurance). This can be attained through the application of policies that match the intent of the data subject to the data, while also enabling the data subject to apply data minimization to

certain data (e.g. sensitive data the data subject is not comfortable sharing, or data deemed not to be relevant for the purpose of service contextualization) prior to its transfer to the service provider. Data subjects are able to opt-in/out of secondary/tertiary processing of the data beyond the original agreed-upon purpose and the transfer to third party organisations, as per their rights under the GDPR.

As in the previous use case, sticky policies, secure enclaves, and the RestAssured toolkit play a central role in simplifying what is required by developers to implement and deploy SGX-based applications.

From a technical perspective, a first draft of a prototype architecture for RestAssured has been developed. This architectural blueprint defines the functionality of all major system components, and defines the high-level interfaces between them.


Acknowledgement. The research leading to these results has received funding from the European Community's Horizon 2020 research and innovation programme under grant agreement no 731678.

References

1. Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I.: Opaque: an oblivious and encrypted distributed analytics platform. In: Proceedings of the 14th USENIX symposium on Networked Systems Design and Implementation (NSDI 2017), pp. 283–298. USENIX Assoc (2017)
2. Mann, Z.Á., Metzger, A.: Optimized cloud deployment of multi-tenant software considering data protection concerns. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017), pp. 609–618. IEEE Press (2017)
3. Dräxler, S., Karl, H., Mann, Z.Á.: Joint optimization of scaling and placement of virtual network services. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017), pp. 365–370. IEEE Press (2017)
4. Schoenen, S., Mann, Z.Á., Metzger, A.: Using risk patterns to identify violations of data protection policies in cloud systems. In: 13th International Workshop on Engineering Service-Oriented Applications and Cloud Services (WESOACS) (2017)
5. Beckers, K., Côté, I., Goeke, L., Güler, S., Heisel, M.: A structured method for security requirements elicitation concerning the cloud computing domain. *Int. J. Secure Softw. Eng. (IJSSE)* 5(2), 20–43 (2014)
6. SurrIDGE, M., Nasser, B., Chen, X., Chakravarthy, A., Melas, P.: Run-time risk management in adaptive ICT systems. In: 8th International Conference on Availability, Reliability and Security (ARES), pp. 102–110. IEEE (2013)
7. Chakravarthy, A., Wiegand, S., Chen, X., Nasser, B., SurrIDGE, M.: Trustworthy systems design using semantic risk modelling. In: Proceedings of the 1st International Conference on Cyber Security for Sustainable Society, pp. 49–81. Digital Economy Sustainable Society Network (2015)
8. ISO/IEC 27001:2013. Information technology – Security Techniques – Information security management systems – Requirements, International Organization for Standardization (2013)



DITAS: Unleashing the Potential of Fog Computing to Improve Data-Intensive Applications

Pierluigi Plebani¹ , David Garcia-Perez², Maya Anderson⁴, David Bermbach³, Cinzia Cappiello¹, Ronen I. Kat⁴, Achilleas Marinakis⁵, Vrettos Moulos⁵, Frank Pallas³, Barbara Pernici¹, Stefan Tai³, and Monica Vitali¹

¹ Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milan, Italy
{pierluigi.plebani,cinzia.cappiello,barbara.pernici,monica.vitali}@polimi.it

² Atos Spain SA, Pere IV, 08018 Barcelona, Spain
david.garciaperez@atos.net

³ Information Systems Engineering Research Group, TU Berlin, Einsteinufer 17, 10587 Berlin, Germany
{db,fp,st}@ise.tu-berlin.de

⁴ IBM Research – Haifa, Haifa University Campus, Mount Carmel, 3498825 Haifa, Israel
{mayaa,ronenkat}@il.ibm.com

⁵ NTUA - National Technical University of Athens, 9 Iroon Polytechniou Street, Zografou Campus, 15773 Athens, Greece
{achmarin,vrettos}@mail.ntua.gr

1 Introduction

Although it has been initially introduced in the telecommunication domain by Cisco [1], Fog Computing is recently emerging as a hot topic also in the software domain, and especially for data-intensive applications (DIA), with the goal of creating a continuum between the resources living on the Cloud and the ones living on the Edge [3]. In fact, especially because of the significant increasing of smart devices connected to the Internet (e.g., smartphones, raspberry PI), operators at the edge of the network are no longer considered as content consumers but also content providers, i.e., the so called *prosumers*. This new scenario implies a paradigm shift and the Fog Computing is contributing to it, by considering Cloud and Edge parts of the same platform.

Among the new arising research challenges [5] that this new paradigm has to deal with, the balance between the different quality of service provided by applications running on the cloud and on the edge becomes fundamental. In fact, services relying on resources exclusively running on the Cloud can be considered scalable and reliable by definition. Conversely, services which are based on resources living on the Edge have the advantage of being very close to the data generators (e.g., sensors and smart devices) thus, the latency associated to the data transmission can be significantly reduced (Fig. 1).

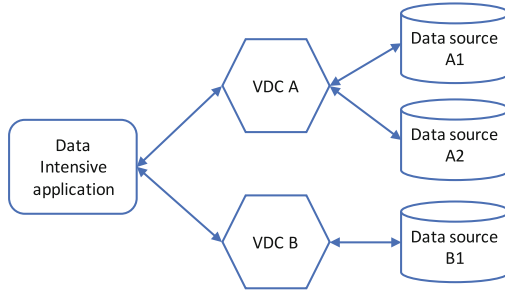


Fig. 1. Data-intensive application model

According to this scenario, the DITAS project¹ aims to provide a Cloud platform composed of a Software Development Kit (SDK) and an Execution Environment (EE) to [4]: (i) simplify the design and the development of DIA in a Fog environment where the advantages of both the sides can be exploited, (ii) improve the execution of the developed applications by enabling both computation and data movement so that data and tasks could migrate from resources on the Cloud to the ones on the Edge – or vice-versa – to satisfy constraints posed by the application designer in terms of performance, security, and privacy.

As data can be distributed among resources both on the Cloud and the Edge, *Virtual Data Containers* (VDCs) are proposed as a mean for timely and securely offering data also transparently with respect to their location and format. Then, the DITAS platform will provide mechanisms enabling the data and computation movement to enable the VDCs to satisfy the DIA requirements. Decision about how, where, and when to move data and computation in DITAS is mainly driven by the analysis of the *data utility*: i.e., the relevance of data for the usage context, where the context is defined in terms of the designer’s goals and system characteristics.

The rest of the paper introduces the current achievements of DITAS project: a DIA model based on VDCs (see Sect. 2) and an architecture which defines the scopes and the roles of the SDK and the EE (see Sect. 3).

2 DITAS Data-Intensive Application Model

The development of a DIA usually requires to take care of where to store data, in which format, how to satisfy the security constraints, and many other aspects which could distract the attention of developers from the business logic. This situation becomes even more clear when dealing with an heterogeneous system where different devices are involved in the data management, as in a Fog environment. This implies that the developers have to manage this heterogeneity, as well as to properly distribute the data among the Edge and the Cloud, to make the application as efficient as possible.

¹ <http://www.ditas-project.eu>.

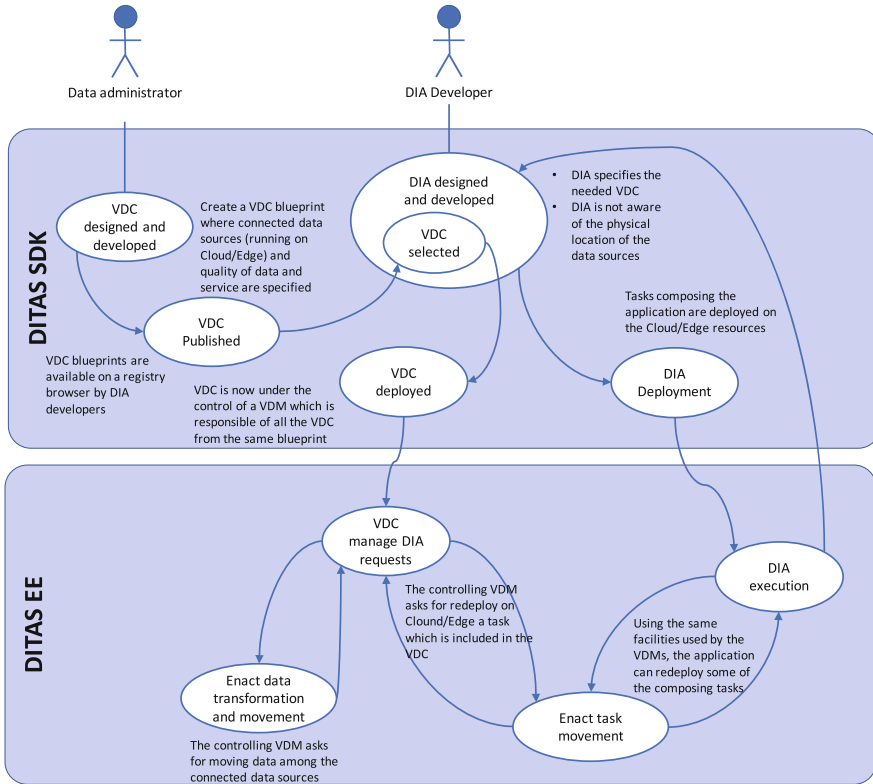


Fig. 2. Data-intensive application lifecycle in DITAS

As shown in Fig. 2, by introducing the VDC, a synergy between the DIA and the data source life-cycle is created. First of all, data administrator defines and publishes a VDC Blueprint to a repository, to make the DIA developers aware of it. Developers design and develop their applications assuming that the data required can be retrieved from a VDC, thus, without knowing how and where the data are stored, but only posing some requirements in terms of quality of data (e.g., accuracy, timeliness) and quality of service (e.g., transmission rate, encryption). A DIA and the related VDCs are now connected each other, so the deployment involve both.

Based on this model, DIA in DITAS are not directly connected to the data sources, but the access to these sources are mediated by VDCs, which provide the following capabilities: (i) a uniform access to data sources regardless of where they run, i.e., on the Edge or on the Cloud; (ii) a set of data processing techniques able to transform data (e.g., encryption, compression); (iii) the possibility to compose these processing techniques in pipelines (inspired by the node-RED programming model²) and to execute the resulting application;

² <http://www.nodered.org>.

(iv) the enactment of data and task movement strategies based on the decision taken by the Virtual Data Manager (VDM), which controls all the VDCs instantiated from the same VDC Blueprint.

3 DITAS Architecture

To support the described DIA model, DITAS is developing: (i) an SDK which will be used to assist both data administrators and DIA application developers, and (ii) an execution environment (EE) where the deployed VDCs and DIAs operate. The main components of these two tools are shown in Fig. 3.

About the SDK, the *VDC Blueprint Editor* helps the data administrator in defining the characteristics of a VDC. The resulting VDC Blueprint contains, among the others, all the information about where the data sources are located, how to access to them, and the interface exposed to the application. As a VDC can internally implement some data processing, the specification of this processing (which adopts the node-red model) is also included in the blueprint. Finally, non-functional properties are specified to define the data utility [2] which includes the reputation, data quality, and QoS dimensions. Given a VDC Blueprint, the EE will have all the information to instantiate a VDC able to manage the underlying data sources while satisfying the specified quality properties.

A *VDC Blueprint repository* makes all the blueprints available to the DIA developers which can submit their functional and non-functional requirements to obtain a list of possible candidates. Then, the developers will select the most

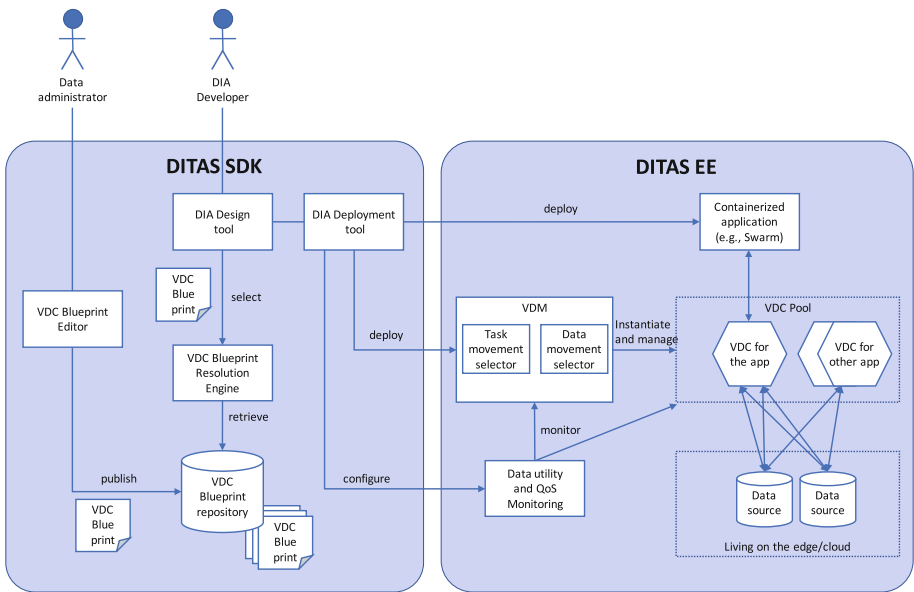


Fig. 3. Data-intensive application lifecycle in DITAS

suitable ones and configure the DIA to be able to be connected to the resulting VDCs at run-time. Based on this approach, the DIA developer does not directly access, and s/he could even ignore, the physical data sources as the data sources from the DIA perspective are the selected VDCs.

The resulting DIA specification, which includes both the application model and the selected VDC Blueprint, is submitted to the *DIA Deployment tool* which represents the bridge between the SDK and the EE as it configures the modules of the EE to host and run the DIA. More specifically, as the application logic of the DIA is based on containerized modules, an orchestration engine for containers (like Docker Swarm or Kubernetes³) will be adopted in the EE to run the resulting system. On the other side, the deployment of the VDC implies the involvement of the VDM, which is in charge of supervising the execution of all the VDC generated from the same VDC Blueprint. For this reason, when a VDC Blueprint is requested for the very first time, a VDM is instantiated and, in turn, instantiates and executes the related VDC. Once another application requires a VDC from the same blueprint, the existing VDM instance will take care of instantiating and controlling also the new VDC. As a consequence, a VDM is in charge of improving the access to the physical data sources connected to the controlled VDCs and it is the component that decides if, how, and when data can be moved among the controlled data sources with the aim of satisfying the requirements posed by all the applications that are connected to the controlled VDCs. Such decisions are taken by the VDM based on the information collected by the *Data Utility and QoS Monitoring* module, which transparently monitors the traffic between the VDCs and the data sources (e.g., what information are requested, the transmission rate).

Acknowledgments. DITAS project is funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement RIA 731945.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16 (2012)
2. Cappiello, C., Pernici, B., Plebani, P., Vitali, M.: Utility-driven data management for data-intensive applications in fog environments. In: de Cesare, S., Frank, U. (eds.) ER 2017. LNCS, vol. 10651, pp. 216–226. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70625-2_20
3. OpenFog Consortium Architecture Working Group: OpenFog Architecture Overview, February 2016. <http://www.openfogconsortium.org/ra>
4. Plebani, P., García-Pérez, D., Anderson, M., Bermbach, D., Cappiello, C., Kat, R.I., Pallas, F., Pernici, B., Tai, S., Vitali, M.: Information logistics and fog computing: the DITAS* approach. In: Proceedings of the CAiSE Forum, Essen, Germany, June 2017
5. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. *IEEE Internet Things J.* **3**(5), 637–646 (2016)

³ <https://docs.docker.com/engine/swarm/>; <https://kubernetes.io>.



HyVar

Scalable Hybrid Variability for Distributed Evolving Software Systems

Thomas Brox Røst¹✉, Christoph Seidl², Ingrid Chieh Yu³,
Ferruccio Damiani⁴ , Einar Broch Johnsen³ , and Cristina Chesta⁵

¹ Atbrox AS, Trondheim, Norway
thomas@atbrox.com

² Technische Universität Braunschweig, Braunschweig, Germany
c.seidl@tu-braunschweig.de

³ Universitetet i Oslo, Oslo, Norway
{ingridcy, einarj}@ifi.uio.no

⁴ Università di Torino, Turin, Italy
damiani@di.unito.it

⁵ Santer Reply SpA, Turin, Italy
c.chesta@reply.it

Abstract. The HyVar project (www.hyvar-project.eu/) proposes a development framework for continuous and individualized evolution of distributed software applications running on remote devices in heterogeneous environments, focusing on the automotive domain. The framework combines variability modeling and software reuse from software product lines with formal methods and software upgrades and can be integrated in existing software development processes. HyVar's objectives are: (O1) To develop a Domain Specific Variability Language (DSVL) and tool chain to support software variability for highly distributed applications; (O2) to develop a cloud infrastructure that exploits software variability as described in the DSVL to track the software configurations deployed on remote devices and to enable (i) the collection of data from the devices to monitor their behavior; and (ii) secure and efficient customized updates; (O3) to develop a technology for over-the-air updates of distributed applications, which enables continuous software evolution after deployment on complex remote devices that incorporate a system of systems; and (O4) to test HyVar's approach as described in the above objectives in an industry-led demonstrator to assess in quantifiable ways its benefits. The end of the project is approaching and we are close to reaching all the objectives. In this paper, we present the integrated tool chain, which combines formal reuse through software product lines with commonly used industrial practices, and supports the development and deployment of individualized software adaptations. We also describe the main benefits for the stakeholders involved.

The HyVar project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 644298.

Keywords: Software engineering · Software maintenance · Software evolution
Software product lines · Variability models · Distributed software
Over-the-air updates · Data intensive systems · Internet of things
Cloud computing

1 Motivation and Approach

1.1 Software Evolution in the Automotive Domain

Evolution is a well-known problem in any software product family of non-trivial complexity. Over the lifespan of a product line, new features are added, old features are deprecated, and separate code branches may be created to deal with spin-off products. As the code and customer base grow, the possible feature combinations used by various product instances soon become unwieldy.

In the automotive domain, all the possible variants of a car (make, model, base equipment, extras etc.) can create a situation with a combinatorial explosion of feature combinations. This poses a challenge when maintaining the supporting software. Not only new features must be catered for but also all previous feature combinations for products that are still on the market and under support agreements. This can easily lead to bad software development practices, such as “clone and own”, where code is copied between repository branches and modified in isolation.

Regarding deployment of software updates in the automotive scenario, how does one ensure that a given car gets an update that is customized to its particular feature combination? Moreover, how can we also take, e.g., the driving characteristics of the car’s owner into account when updating the car’s software? Using traditional software development techniques, it is easy to end up with a software repository with lots of duplicated and overlapping code, where making any substantial change carries the risk of unforeseen down-the-line implications.

1.2 HyVar Solution

Within the HyVar project, we take a two-fold approach towards tackling this problem, focusing on both the *development* and *deployment* aspect of maintaining complex software products.

On the *development* side, we have created a DSL and other tools that use software product line (SPL) modeling techniques for a structured approach towards handling feature combinations [1–3]. These tools can either be adopted for development from scratch for new projects or be applied to existing projects with low adoption efforts. Moreover, our analysis tools can be applied to a project that is already suffering from previous clone-and-own development so that differences and similarities of cloned products are automatically extracted and modeled [4]. This greatly simplifies the task of cleaning up from years of bad development practices.

On the *deployment* side, we introduce a cloud-based tool chain that complements our development tools [5]. The tool chain functions in an automotive context, keeping tabs on a large number of connected devices and their feature configurations. A re-configuration component detects whenever a software update must be applied, either

due to changes on the device (pull action) or changes on the software side (push action). Instead of taking the naïve approach of pushing the same monolithic update to each device, the tool chain creates a customized update for each device and only does the full recompile when it is deemed necessary. This greatly reduces the complexity and bandwidth required to do over-the-air updates for a device fleet. Also, as part of the supporting toolkit, we can do a static analysis of the resources required for running our tool chain on the cloud. For a realistic traffic pattern model, this removes a lot of the guesswork otherwise necessary for cloud infrastructure cost estimation.

2 Application and Benefits

The approach has been applied to an automotive domain scenario, namely the dynamic reconfiguration of car software based on context, where we addressed the following issues: (i) To develop a software product line by allowing developers to derive a new product from an existing one; (ii) To reduce the risk in distributed software development projects by developing several software product lines with distributed teams keeping track of the dependencies; (iii) To support personalized deployment from the cloud; and (iv) To derive a software product line from existing products. For the stakeholders in our automotive scenario (e.g., car manufacturers and automotive software developers), there are several benefits associated with using our tool chain. Some of these benefits are outlined below, grouped according to issues (i)–(iv).

2.1 Software Product Line Development Using the HyVar Tool Chain

We experienced the following benefits by developing a software product line realizing the emergency call service for both the European and Russian market, exploiting the commonalities and ensuring compliance with the respective standards.

The existing product can be used as it is. Using the DSVL and delta modeling techniques to transform statecharts and/or source code, it is possible to start from an existing product.

New features are fully implemented and recorded in statecharts. The configuration differences between product branches are highly visible and explicit so that they are much easier to communicate within the company. Moreover, the executable programs can be created directly from the statechart editor.

Living models. As new features are both modeled and built from statecharts, there is a direct connection between the model and the final product.

Reduced code duplication and development time. As code is generated from models, the amount of code duplication is reduced. The copy/paste approach towards software development is no longer needed.

2.2 Reducing Risk in Distributed Software Development Projects

Our initial demonstrator involved a single software product line. We then extended it into a more complex system, including three software product lines with interdependencies. This yielded additional benefits.

All interfaces are defined through MSPL feature model interfaces. This encourages both better encapsulation and a more structured approach towards feature interfaces and simplifies the distributed development.

Independent software product line. By the use of feature model interfaces of the HyVar tool chain, the new functionality can be planned and constructed independently from the rest of the software product line.

Feature encapsulation helps evolution. Better encapsulation makes it explicit which parts of the software system can be changed without introducing errors in the existing functionality.

Early detection of specification errors. Using feature model interfaces, it is possible to guarantee that only intended configurations can be created.

2.3 Personalized Deployment from the Cloud

In our final demonstrator we enabled software updates that take the driving style and preferences of individual drivers into account.

Cloud-based infrastructure. One of the major benefits of using cloud services is that you only pay for the resources you use. This means that there is no need for an upfront data center investment and that the costs scale along with the number of users as the company grows. For customers who are wary of using public clouds, private cloud installations are also possible.

Simulation model of cloud resource requirements. With the traffic data collected by a car manufacturer stakeholder, it is possible to simulate the resources needed for the tool chain cloud infrastructure. This makes it possible to estimate the costs required for deploying the tool chain for a given fleet of cars even if there are peak periods.

Context changes can be reflected in the software configuration. Using validity formulas, context constraints and the HyVarRec reconfigurator, the software can be customized for a highly specific environment.

2.4 Derivation of an SPL from Existing Products

We have developed a variability mining methodology that provides the benefits listed below. Although the methodology is compatible with the HyVar tool chain, the HyVar demonstrator is not suitable for evaluating the methodology. Therefore, we have evaluated it by considering another case study in the automotive domain.

Automated analysis of existing software products. The differences and similarities of existing, cloned products can be analyzed almost completely automatically. From this, it is possible to generate feature models, mappings and delta modules that later can be used when adding new features or spinning off new product lines. The effort compared to doing this manually is reduced greatly.

Generated software product line elements. From the results of the analyses, the variability mining also generates suitable elements, such as delta modules, a technical feature model and, if needed, even an entire suitable delta language, e.g., for elements written in domain-specific languages. This overall greatly reduces the manual effort to set up an SPL from cloned variants.

Controlled restructuring. The process is not fully autonomous, meaning that developers have a lot of control over things such as feature naming and how the mined products should be restructured. They can also guide the restructuring through doing an iterative feedback/adaption process with the variability mining technology.

Increased abstraction between features and code. As the feature models are refined, so is the abstraction to the underlying code. This has long-term benefits in terms of both planning, discussing and working with a product as a combination of evolved features rather than just lines of code.

3 Conclusion

We have presented an innovative solution to the software reuse problem, integrating SPL engineering principles with existing tools and commonly used industrial practices. The HyVar approach supports the development and deployment of individualized software adaptations and realizes the concept of hybrid variability. The methodology and tool chain has been applied in a scenario from the automotive domain, and seems promising also for other emerging scenarios, such as Internet of Things (IoT) and Cyber-Physical Systems (CPS), characterized by a huge number of remote devices, each of which has its own hardware configuration, runs a customizable distributed software application and needs to evolve in order to fix or prevent misbehavior, to adapt to environmental changes, accomplish new regulations, satisfy new user requests or meet new market expectations.

References

1. Chesta, C., Damiani, F., Dobriakova, L., Guernieri, M., Martini, S., Nieke, M., Rodrigues, V., Schuster, S.: A toolchain for delta-oriented modeling of software product lines. In: Margaria, T., Steffen, B. (eds.) ISO/FA 2016. LNCS, vol. 9953, pp. 497–511. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_40
2. Nieke, N., Engel, G., Seidl, C.: DarwinSPL: an integrated tool suite for modeling evolving context-aware software product lines. In: ter Beek, M.H., Siegmund, N., Schaefer, I. (eds.) Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems (VAMOS 2017), pp. 92–99. ACM (2017). <https://doi.org/10.1145/3023956.3023962>
3. Damiani, F., Lienhardt, M., Paolini, L.: A formal model for multi SPLs. In: Dastani, M., Sirjani, M. (eds.) FSEN 2017. LNCS, vol. 10522, pp. 67–83. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68972-2_5
4. Wille, D., Schulze, S., Seidl, C., Schaefer, I.: Custom-tailored variability mining for block-based languages. In: Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016). IEEE (2016). <https://doi.org/10.1109/saner.2016.13>
5. Mauro, J., Nieke, N., Seidl, C., Chieh Yu, I.: Context aware reconfiguration in software product lines. In: Schaefer, I., Alves, V., de Almeida, E.S. (eds.) Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2016), pp. 41–48. ACM (2016). <https://doi.org/10.1145/2866614.2866620>



Enhancing Big Data Application Design with the DICE Framework

Giuliano Casale and Chen Li^(✉)

Imperial College London, London, UK
{g.casale, chen.li1}@imperial.ac.uk

Abstract. The focus of the DICE project is to define a quality-driven framework for developing Big data applications. DICE offers an Eclipse-based development environment, centered around a novel UML profile, to prototype, deploy, monitor, and test Big data applications. The DICE framework has been designed to natively support popular open-source solutions. The framework offers a set of 15 open source tools, which have been validated against industrial case studies in the news and media, port operations, and e-government domains.

Keywords: Quality · Big data · UML · Eclipse · DevOps

1 Overview

DevOps has become mainstream in recent years as a movement that attempts to lower the barrier between IT development and operation teams in order to increase the agility of the application release process. One requirement to implement this successfully is to share a unified set of concepts, models, and tools among developers and operators. Even though DevOps tools are rapidly growing in the industry, few of them are natively designed to support Big data applications, even though these have gained much traction in the industry in recent years. By Big data application, we here mean applications that rely on core Big data processing technologies such as Storm, Cassandra, Hadoop/MapReduce, Spark, MongoDB, and many others.

The main objective of the DICE framework is to deliver a quality-driven DevOps toolchain for Big data applications that natively support these Big data technologies. The DICE architecture is shown in Fig. 1 and is centered around the following main components:

- **DICE IDE:** this is an Eclipse-based IDE built upon a new UML profile that abstracts the main characteristics of open source Big data technologies.

C. Li—This paper has been supported by the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 644869. Project full name: DICE - Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements: Feb 2015–2018, website: www.dice-h2020.eu.

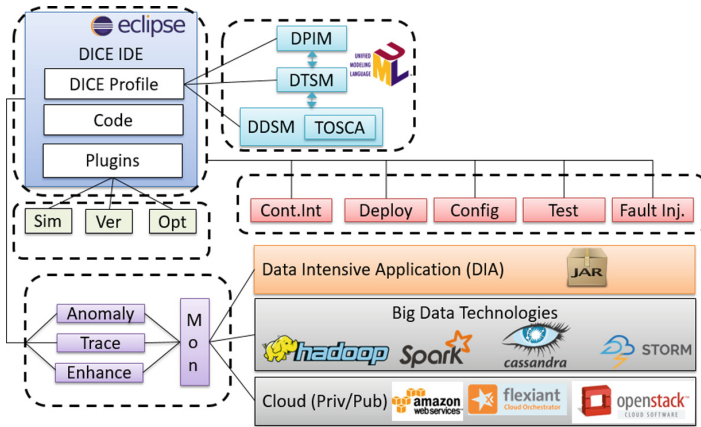


Fig. 1. DICE framework - available at <http://github.com/dice-project/>.

For example, with the DICE UML profile, a developer can express requirements on the number and topology of spouts and bolts in a Storm-based application, and associate them to specific virtual machine instances in the cloud. Moreover, the Papyrus-based UML diagram visualization allows to iteratively refine the application design using a methodological workflow integrated within the IDE as Eclipse cheat-sheets.

- **Quality analysis plugins:** these tools include discrete-event simulation and verification of the application architecture in the early design stages. These tools allow in particular to predict what latency will be experienced by the users and verify quality guarantees offered by the design. The plugins also include an architecture optimization tool that decides the resources to be assigned upon deployment to the Big data application (e.g., number and type of VMs).
- **Continuous delivery and testing tools:** these tools take care of continuous integration, deployment, configuration, load testing, and fault injection of the application in the runtime environment. The deployment tool, in particular, is TOSCA-compliant and can be activated directly from within the DICE IDE, allowing developers to reduce the time they need to deploy a Big data application prototype to the cloud. Supported cloud environments include AWS, OpenStack, and Flexiant FCO.
- **Monitoring and feedback analysis tools:** these tools collect monitoring data from the running application and analyze it by means of machine learning-based anomaly detection, trace checking, and statistical inference techniques. Altogether, they allow examining whether the application is behaving as expected in the runtime environment. Moreover, these tools supply monitoring information to the development environment, to guide future iterations on the prototypes.

The above capabilities have been validated on industrial demonstrators concerning processing news streams from Twitter, cloud-based software systems for port operations, and Big data applications for tax fraud detection. The outcome of these validations is that an integrated toolchain can substantially accelerate the design and testing of Big data applications, without requiring steep learning curves.

2 The DICE Enhancement Tool

In this section, we illustrate a specific result of the DICE project, the *DICE Enhancement tool*. The goal of this tool is to provide feedback to DICE developers on the application behaviour at runtime, leveraging the monitoring data collected from the DICE Monitoring Platform (DMon), in order to help them iteratively enhance the application design. The DICE Enhancement tools include two modules: the DICE Filling-the-Gap (DICE-FG), which helps focusing on statistical estimation of UML parameters used by the quality analysis plugins for simulation and optimization, and DICE-APR (Anti-Patterns and Refactoring), a tool that detects anti-patterns in the UML design and supplies recommendation on how to resolve them.

The methodological workflow of this tool is shown in Fig. 2. DICE-FG provides a performance and reliability analysis of big data applications and updates UML models with analysis results. DICE-APR transforms the DICE UML model annotated with DICE profiles into a Layered Queueing Network (LQN), which is a queueing model that can be used for performance prediction. For example, an LQN can predict the latency expected for the application under a given workload. The results are predicted CPU utilizations and latencies at the resources, which will be used for APs detection and for recommending refactoring decisions.

DICE-FG is designed to achieve the following objectives:

- Provide statistical estimation algorithms to infer resource consumption of an application.
- Provide fitting algorithms to match monitoring data to parametric statistics distributions.

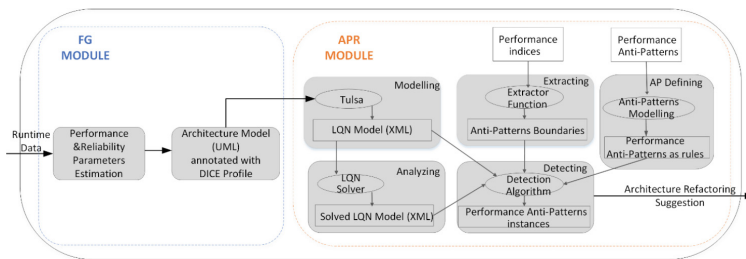


Fig. 2. DICE enhancement tool workflow.

- Use the above algorithms to parameterize UML models annotated with the DICE profile.
- Acquire data via JSON and the DICE Monitoring platform (DMon).

The main logical components of the DICE-FG tool are the Analyzer and the Actuator.

- **DICE-FG Analyzer:** The DICE-FG Analyzer executes the statistical methods necessary to obtain the estimates of the performance models parameters, relying on the monitoring information available on the input files.
- **DICE-FG Actuator:** The DICE-FG Actuator updates the parameters in the UML models, e.g., resource demands, which are obtained from the DICE-FG Analyzer.

The DICE-APR module is designed to achieve the following objectives:

- Transforming UML diagrams annotated with DICE profiles to performance model (i.e., Layered Queueing Networks) for performance analysis.
- Specifying the selected popular AP of DIAs (e.g., defining AP rules in executable codes).
- Detecting the potential AP from the performance model.
- Generating refactoring decisions to update the architecture model to fix the design flaws according to the AP solution.

The components of the DICE-APR module are Model-to-Model Transformation, Anti-patterns Detection and Architecture Refactoring as detailed below.

- **Model-to-Model Transformation:** It provides the transformation of annotated UML model with DICE Profile into quality analysis model. The target performance models is Layered Queueing Networks.
- **Anti-patterns Detection:** The Anti-patterns detection component relies on the analysis results of the Model-to-Model Transformation component. The selected anti-patterns are formally specified for identifying if there are any anti-patterns issues in the model.
- **Architecture Refactoring:** According to the solution of discovered anti-patterns, refactoring decisions will be proposed, e.g., component reassignment, to solve them. The architecture model will be shared back to the DICE IDE for presentation, to the user in order to decide if the proposed modification should be applied or not.

The implementation of the model-to-model transformations, from DICE UML to LQN, used by DICE-APR rely on a tool called Tulsa [2], which is based on the Epsilon language (i.e., Epsilon Transformation Language (ETL) and Epsilon Object Language (EOL)). The LINE solver¹ is then used to compute performance metrics from the LQN model generated by Tulsa. The APs detection is implemented using Matlab scripts that iteratively call the LINE API to determine if a specific refactoring action will yield a better latency or resolve CPU bottlenecks.

¹ LINE website: <http://line-solver.sf.net>.

We have applied the DICE APR tool to case studies of the Wikistats open source application, which is based on Apache Storm, finding that it can effectively detect and resolve two common antipatterns: Infinite Wait (IW) and Excessive Calculation (EC):

- IW occurs when a component must call several servers to complete the task. If a large amount of time is required for each service, performance will suffer. The solution is to report the component that causes the IW and provide component replication or redesign suggestions to the developer.
- EC occurs when a processor performs all of the work of an application or holds all of the application data. This anti-pattern manifests itself as an excessive amount of CPU calculations that degrade performance. The solution is to report to the developer the processor that causes the EC and suggest to add a new processor to migrate tasks to the developer.

We point to [3] for details and experimental results of the DICE APR tool.

A pre-requisite for the APR tool to generate meaningful predictions is to know how many resources each software components requires to carry out its operation, under a given workload demand. The DICE FG tool delivers this capability, based on a number of statistical estimators that transform monitoring metrics such as throughputs and utilizations into CPU requirements for the application. DICE-FG uses a collection of algorithms, ranging from regression methods to statistical inference techniques over queueing networks. We have recently reported on the relative accuracy of these methods in estimating resource demands [4], finding that no single method dominates the others, thus a module like FG that offers multiple estimation algorithms allows to adapt the situation to the different situations that arise in practice.

3 Conclusion





The DICE IDE and updates on the project are available at <http://www.dice-h2020.eu>, together with a project vision document [1]. In the future we plan to release specialized “thinned” product versions of the IDE dedicated to specific technologies, such as Apache Storm, in order to supply to end users an environment targeting specific application classes (e.g., stream-based applications).

References

1. Casale, G., et al.: DICE: quality-driven development of data-intensive cloud applications. In: Proceedings of MiSE Workshop (2015)
2. Li, C., Altamimi, T., Zargari, M.H., Casale, G., Petriu, D.: Tulsa: a tool for transforming UML to layered queueing networks for performance analysis of data intensive applications. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 295–299. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66335-7_18
3. Li, C., Casale, G.: Performance-aware refactoring of cloud-based big data applications. In: Proceedings of CSCI-ISCC (2017)
4. Spinner, S., Casale, G., Brosig, F., Kounev, S.: Evaluating approaches to resource demand estimation. *Perform. Eval.* **92**, 51–71 (2015)



Developing, Provisioning and Controlling Time Critical Applications in Cloud

Zhiming Zhao¹ , Paul Martin¹ , Andrew Jones² , Ian Taylor²,
Vlado Stankovski³ , Guadalupe Flores Salado⁴, George Suci⁵,
Alexandre Ulisses⁶, and Cees de Laat¹

¹ University of Amsterdam, Science Park 904,
1098XH Amsterdam, The Netherlands
z.zhao@uva.nl

² Cardiff University, Queen's Buildings, 5 The Parade, Cardiff CF24 3AA, UK

³ University of Ljubljana, Ljubljana, Slovenia

⁴ Wellness Telecom SL, Seville, Spain

⁵ BEIA Consult International SRL, Bucharest, Romania

⁶ MOG Technologies SA, Maia, Portugal

Abstract. Quality constraints on time critical applications require high-performance supporting infrastructure and sophisticated optimisation mechanisms for developing and integrating system components. The lack of software development tools and in particular cloud-oriented programming and control models make the development and operation of time critical cloud applications difficult and costly. The SWITCH project (Software Workbench for Interactive, Time Critical and Highly self-adaptive Cloud applications) addresses the urgent industrial need for developing and executing time critical applications in Clouds. The primary users of SWITCH are Cloud application developers who wish to design and develop elastic, time-critical applications for the federated Cloud. By using SWITCH and its services they can discover appropriate infrastructures, choreograph their applications and QoS/QoE dependencies, and configure their applications for execution. They can choose where to deploy these applications using a specific target infrastructure (e.g. an appropriately selected Cloud provider). They can also manage and monitor their running applications so that they are always running optimally.

Keywords: Time critical applications · Cloud · Quality of user experience
Infrastructure programming · Self-adapting systems

1 Introduction

Quality constraints on applications, such as limiting network latency and jitter in live event broadcasting or reducing processing delay for real-time sensor data in disaster early warning systems, require high performance supporting infrastructure, along with sophisticated optimization mechanisms for developing and integrating system components. Cloud environments provide virtualized, elastic, and controllable on-demand

services for supporting complex application systems. The lack of software development tools and in particular cloud-oriented programming and control models make the development and operation of time critical cloud applications difficult and costly. A software workbench, which can couple the development, deployment and operation phases of the time critical application lifecycle within cloud environments, will reduce the development complexity of building such applications and improve the efficiency of runtime application control [4, 5].

In the EU H2020 project SWITCH (the Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications), three highly customisable subsystems are proposed for the application development, virtual infrastructure customisation and provisioning, and runtime adaptation respectively. In this paper, we first discuss the key technical challenges arising in the development of the workbench based on requirements analysis and technical review, and then present the current achievements in the development of the three subsystems.

2 Time Critical Application Cases and Requirements

The SWITCH project focuses on three use cases: *collaborative real-time business communication*, *elastic disaster early warning*, and *cloud studio for directing and broadcasting live events*. In each of these examples we can see different time related quality attributes, e.g., communication latency, data processing and decision-making time, and response time for switching between video streams. The ability to keep these attributes within certain bounds clearly determines the delivery of the business value of the cases, e.g., an early warning system cannot be effective when decisions cannot be made within a certain time window, a business collaboration won't meet business needs if the user experience is destroyed by low audio/video quality caused by latency, and a video studio application won't be accepted by any professional broadcasting station if the video switching response time is not guaranteed. We classify different time critical constraints as speed critical (e.g., latency sensitive), real-time (focusing on timeliness) and nearly real-time [6].

To meet these different types of time critical constraint in cloud, both application and virtual infrastructures (i.e. networked virtual machines hosting the application) have to be customized during the design phase, and controllable at runtime to allow adjustment performance as needed. We can highlight different steps in the application lifecycle via three key overlapped cycles of *application development*, *provisioning* and *runtime* (see Fig. 1).

The SWITCH workbench is proposed specifically to address each one of these issues in turn within a single integrated toolset.

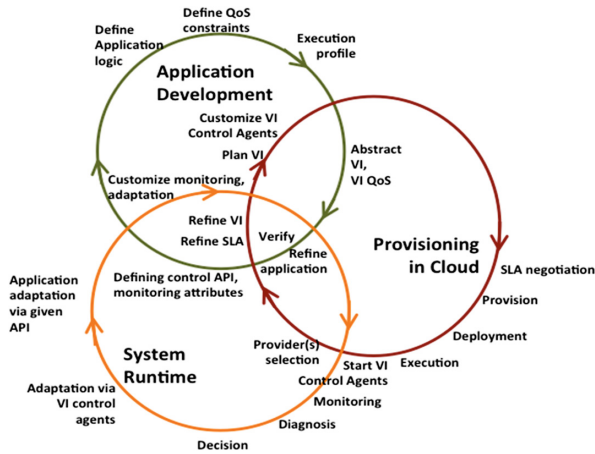


Fig. 1. The key phases of time critical Cloud applications.

3 The SWITCH Approach

The core idea of the SWITCH environment is a new development and execution model on both application and infrastructure for time critical cloud applications. The new model brings together application composition, execution environment customisation, and runtime control, which are normally treated as separate processes, into one optimisation loop based on time critical requirements. The SWITCH environment employs an ontological framework [8] to guide each step in the development, and tools are delivered to the users via three subsystems: SIDE, DRIP and ASAP.

The **SWITCH Interactive Development Environment (SIDE)** subsystem provides interfaces for all of the user- and programmer-facing tools, by exposing a collection of graphical interfaces and APIs that tie in SWITCH's services to a Web-based environment. The **Dynamic Real-time Infrastructure Planner (DRIP)** subsystem prepares the execution of the applications developed in the SIDE subsystem by: (1) identifying the constraints on infrastructure resources required to meet the time-critical requirements of the applications; (2) defining an optimal virtual runtime environment that meets those constraints; (3) provisioning the planned environment with the chosen resource provider; and (4) deploying the components required by the application. The **Autonomous System Adaptation Platform (ASAP)**: (1) monitors the status of the application and the runtime environment; (2) examines the actual performance of the required quality attributes; (3) autonomously controls the application and runtime environment to maintain optimal system level performance against the time critical constraints; and (4) learns from its own decision history to improve its intelligence in making future decisions for autonomous control.

4 Software Workbench for Time Critical Cloud Applications

The software of SWITCH workbench is open source under Apache License 2.0¹. Figure 2 depicts the application-infrastructure composition GUI (the right side) provided by the SIDE subsystem, and its interaction with provisioning and runtime control (the left side). The GUI provides customizable viewpoints of the GUI based on the different kinds of activity engaged in by the developer.

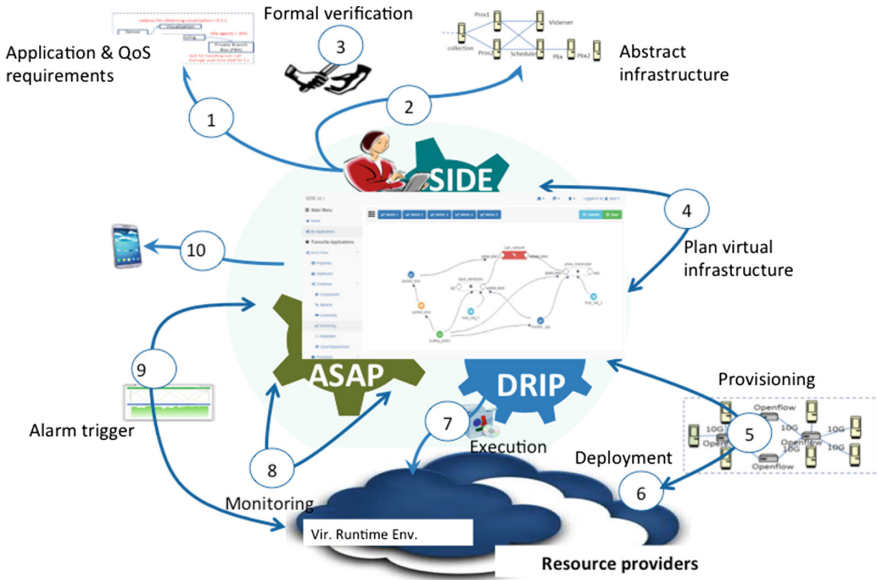


Fig. 2. All subsystems are glued via the GUI of SIDE.

The application developer begins with composing the application logic and defining the QoS constraints, such as the latency for state visualisation, sensor event handling delay (step 1). The developer can also give an abstract network overlay to define the runtime environment (step 2). These activities can be optimised and aided using a formal reasoning component integrated in SIDE (step 3). Currently, TOSCA² is used to describe the logic structure of application-infrastructure.

The results of step 1 and 2 will be passed from SIDE to DRIP; the application logic will be annotated with its time constraints, e.g., deadlines. DRIP will select suitable virtual machine resources from given set of providers to match those time constraints using a partial critical path (PCP) based approach [3, 9] (step 4). The planned virtual infrastructure will then be provisioned on the selected provider(s). A parallel provisioning mechanism is provided based on the mapping between topology and the

¹ SWITCH software repository: <https://github.com/switch-project/SWITCH>.

² <https://www.oasis-open.org/committees/tosca>.

providers [7]. An Open Cloud Computing Interface (OCCI)³ based provisioning interface is supported (step 5). A deadline aware deployment approach is developed in DRIP to deploys application components [1] (step 6). At runtime, SIDE allows the user to: query and visualise the runtime status of the application and runtime environment (step 8) [2], receive notification of system status and inform the user (step 9) [10], directly manipulate the system execution (step 10) [11].

5 Summary

In this paper, we provided an overview of the overall approach in SWITCH for developing time critical applications, and then provisioning and operating them in a cloud-based environment. The workbench has three subsystems for application development, cloud virtual infrastructure provisioning, and runtime operation respectively. These subsystems implement optimisation mechanisms across different layers between applications and virtual infrastructures. In the final phase of the project, three pilot use cases will be prototyped and demonstrated using the full SWITCH workbench.

Acknowledgement. This research has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project).

References

1. Hu, Y., Wang, J., Zhou, H., Martin, P., Taal, A., de Laat, C., Zhao, Z.: Deadline-aware deployment for time critical applications in clouds. In: Proceedings of the Euro-Par (2017)
2. Taherizadeh, S., Jones, A., Taylor, I., Zhao, Z., Martin, P., Stankovski, V.: Runtime network-level monitoring framework in the adaptation of distributed time-critical cloud applications. In: The 22nd International Conference on Parallel and Distributed Processing Techniques and Applications (2016)
3. Wang, J., Taal, A., Martin, P., Hu, Y., Zhou, H., Pang, J., de Laat, C., Zhao, Z.: Planning virtual infrastructures for time critical applications with multiple deadline constraints. *Int. J. Future Gener. Comput. Syst.* **75**, 365–375 (2017)
4. Zhao, Z., Martin, P., Wang, J., Taal, A., Jones, A., Taylor, I., Stankovski, V., Garcia Vega, I., Suci, G., Ulisses, A., de Laat, C.C.: Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach. *Procedia Comput. Sci.* **68**, 17–28 (2015)
5. Zhao, Z., Taal, A., Jones, A., Taylor, I., Stankovski, V., Garcia, I., Jesus, F., Suci, G., Ulisses, A., Ferreira, P., de Laat, C.: A software workbench for interactive, time critical and highly self-adaptive cloud applications (SWITCH). In: The Proceedings of 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2015)

³ <http://occi-wg.org/>.

6. Koulouzis, S., Martin P., Carval, T., Grenier, B., Judeau, G., Wang, J., Zhou, H., de Laat, C., Zhao, Z.: Seamless Infrastructure customisation and performance optimisation for time-critical services in data infrastructures. In: Proceedings of the 8th International Workshop on Data-Intensive Computing in the Clouds, ACM SIGHPC, in IEEE Supercomputing (2017)
7. Zhou, H., Hu Y., Wang, J., Martin, P., de Laat, C., Zhao, Z.: Fast and dynamic resource provisioning for quality critical cloud applications. In: IEEE International Symposium on Real-time Computing (ISORC) (2016)
8. Martin, P., Taal, A., Quevedo, F., Rogers, D., Evans K., Jones, A., Stankovski, V., Taherizadeh, S., Trnkoczy, J., Suciú G., Zhao, Z.: Information modelling and semantic linking for a software workbench for interactive, time critical and self-adaptive cloud applications. In: The Workshop of CCPI-2016, in the Proceedings of the 30th IEEE International Conference on Advanced Information Networking and Applications (AINA) (2016)
9. Wang, J., de Laat, C., Zhao, Z.: QoS-Aware virtual SDN network planning. In: IFIP/IEEE International Symposium on Integrated Network Management, Lisbon, Portugal, 8–12 May 2017
10. Paščinski, U., Trnkoczy, J., Stankovski, V., Cigale, M., Gec, S.: QoS-aware orchestration of network intensive software utilities within software defined data centres. *J. Grid Comput.* **16** (1), 85–112 (2018)
11. Evans, K., Jones, A., Preece, A., Quevedo, F., Rogers, D., Spasić, I., Taylor, I., Stankovski, V., Taherizadeh, S., Trnkoczy, J., Suciú, G., Suciú, V., Martin, P., Wang, J., Zhao, Z.: Dynamically reconfigurable workflows for time-critical applications. In: International Workshop on Workflows in Support of Large-Scale Science, in IEEE Supercomputing (2015)



MIKELANGELO: Micro KERnel virtualizAtion for hiGh pERfORMance cLOUD and HPC Systems

Nico Struckmann¹(✉), Yosandra Sandoval¹(✉), Nadav Har'El², Fang Chen³,
Shiqing Fan³, Justin Činkelj⁴, Gregor Berginc⁴, Peter Chronz⁵, Niv Gilboa⁶,
Gabriel Scalosub⁶, Kalman Meth⁷, and John Kennedy⁸

- ¹ High Performance Computing Center Stuttgart (HLRS), University of Stuttgart,
Nobelstr. 19, 70569 Stuttgart, Germany
{struckmann,sandoval}@hlrs.de
- ² ScyllaDB, Ltd., 11 Galgalei Haplada, 4672211 Herzelia, Israel
nyh@scylladb.com
- ³ European Research Center, Huawei Technologies Duesseldorf GmbH,
Riesstr. 25, 80992 Munich, Germany
{fang.chen1,shiqing.fan}@huawei.com
- ⁴ XLAB Razvoj programske opreme in svetovanje, d.o.o. (XLAB),
Pot Za Brdom 100, 1000 Ljubljana, Slovenia
{justin.cinkelj,gregor.berginc}@xlab.si
- ⁵ Gesellschaft fuer wissenschaftliche Datenverarbeitung mbH Goettingen (GWDG),
am Fassberg 11, 37077 Göttingen, Germany
peter.chronz@gwdg.de
- ⁶ Ben-Gurion University of the Negev (BGU),
Office of the President - Main Campus, 84105 Beer Sheva, Israel
{gilboan,sgabriel}@bgu.ac.il
- ⁷ IBM Israel Science and Technology Ltd., 94 Derech em-Hamoshavot,
49527 Petach Tikva, Israel
meth@il.ibm.com
- ⁸ Intel Shannon Limited (INTEL), Collinstown Industrial Park, Leixlip, Ireland
john.m.kennedy@intel.com

Abstract. MIKELANGELO is a project, targeted to disrupt the core underlying technologies of Cloud computing, enabling even bigger uptake of Cloud computing, HPC in the Cloud and Big Data technologies under one umbrella. The vision of it is to improve responsiveness, agility and security of the virtual infrastructure through packaged applications, using lean guest operating system OSv and superfast hypervisor SuperKVM. In short, the work will concentrate on improvement of virtual I/O in KVM, using additional virtio expertise, integrated with the light-weight operating system OSv and with enhanced Security. The HPC in the Cloud focus will be provided through involvement of a large HPC centre, with the ability and business need to cloudify their HPC business. The Consortium consists of hand-picked experts (e.g., the original creator of KVM - Avi Kivity) who participate in the overall effort to reduce one of the last performance hurdles in the virtualisation (I/O). Other layers

of inefficiency are addressed through OSv (thin operating system). Such approach will allow for use of MIKELANGELO stack on heterogeneous infrastructures, with high responsiveness, agility and improved security. The targeted audience are primarily SMEs (e.g. simulation dependent SMEs) and data center operators who either benefit from higher performance or flexibility, introduced by the software stack. Four real world use-cases with clear owners, serve as validators and also directly contribute to the exploitation of project results.

Keywords: Cloud computing · HPC · Big data · SuperKVM · KVM Virtualised infrastructures · OSv · vTorque

1 Introduction

MIKELANGELO project¹ [1,3] is about improved performance, flexibility and manageability of virtualised infrastructures. The project deals with a wide range of technologies on many different levels of a typical stack, such as cloud and HPC. These involve the hypervisor, the guest operating system and the management layers. On top of that, this holistic approach puts MIKELANGELO into a unique position with the capacity of providing several cross-cutting technologies extending the potential of individual components. Today's Cloud and HPC architectures are inefficient, as there is always a trade-off between flexibility, efficiency, stability and security. Cloud environments are in general more flexible than static HPC environments, in terms of operating systems, kernel version, software availability and abstraction of compute environments, i.e. shared file system mount paths. However, Cloud's flexibility provided by virtualization comes with the disadvantage of a high overhead for I/O intensive applications. The overhead of I/O performance in virtual environments when compared to native performance of compute nodes is not suitable for many HPC workloads that are latency-sensitive or have high I/O rates [2]. The project addresses the whole stack comprising typical infrastructure as a service (IaaS) or, with the support of some of the use cases, even platform as a service (PaaS). Its core development focuses on enhancements made to the KVM hypervisor and several, both functional and non-functional, improvements of the guest operating system. The optimized hypervisor sKVM targets, in combination with the lean guest operating system and further optimizations for virtualized I/O, like virtual RDMA (vRDMA), improved I/O core scheduling (IOCM), Zero-Copy transmission (ZeCoRX) and shared memory between VMs running on the same host (UNCLOT), snap for holistic telemetry and SCAM for greater security, to improve the overall performance, flexibility and manageability of the two formerly distinct Cloud and HPC environments. All of these components are transparently integrated into commonly used middlewares, the widely used HPC batch-system PBS/Torque [4] for HPC environments and OpenStack for

¹ <https://www.mikelangelo-project.eu>.

Clouds. vTorque extends PBS/Torque and provides the baseline for the integration of several MIKELANGELO components into HPC environments. It comes with management capabilities by the creation of virtual environments and the deployment of batch workloads in such. While MCM enhance OpenStack by new scheduling capabilities, Scotty provides by continuous integration support. All components can also be deployed independently.

2 MIKELANGELO Architecture

This document describes the current status of the final architecture and provides an overview of all components involved and where they are located in the different layers the MIKELANGELO framework. The architecture is designed to improve the I/O performance in virtualised environments and also to bring the benefits of flexibility through virtualization to HPC systems. These benefits include, besides application packaging and application deployment, also elasticity during the application execution, without losing the high performance in computation and communication provided by HPC infrastructures. Each of the components can be used independently of other, however the benefits sum up combining as much components as possible. The diagram in Fig. 1 shows a high level overview of these components and their relations. The Hypervisor Architecture, sKVM, aims at improving performance and security at the lower layer of the architecture, the hypervisor. Previous work was divided into three separate components: IOcm, an optimization for virtual I/O devices that uses dedicated I/O cores, virtual RDMA (vRDMA) for low overhead communication between virtual machines, UNCLOT for shared memory between VMs running on the same host, and ZeCoRx avoids copying data between host and guest buffers on the receive path. Further SCAM, a security feature identifying and preventing cache side-channel attacks from malicious co-located virtual machines in multi tenant Cloud environments. The guest operating system (or “guest OS”) is the operating system, it implements the various APIs (Application Programming Interfaces) and ABIs (Application Binary Interfaces) which the applications utilize. The goals of enhancing the guest operating system are to run I/O-heavy and HPC (high performance computing) applications more efficiently than on traditional virtualized operating systems. Several cross-layer optimizations are targeted, ranging from hypervisor on host OS up to the guest OS. In addition to improving efficiency, another goal of the project is to simplify deployment of applications in the cloud. MIKELANGELO initially focused only on application package management with its extensions done primarily to the Capstan open source project. A completely new way of composing self-sufficient virtual machine images based on OSv [5] is introduced. It allows for flexible and efficient reuse of pre-built application packages that are readily provided by the consortium. It also builds on best practices on how to approach the package preparation with various packages from the HPC and Big data fields. Since then, progressed towards cloud management and application orchestration. This comprises the integration of full support for deployment of unikernels onto OpenStack. Application orchestration using container-like interfaces is the last step towards management of

lightweight applications on top of heterogeneous infrastructures. All of this has been joined under one main toolbox (LEET - Lightweight Execution Environment Toolbox). On top of these components, the project also delivers components spanning over all the layers of the software stack. This includes snap and SCAM. Snap is a holistic telemetry solution aimed at gathering data from all the layers of the target infrastructures, ranging from the hardware through all layers of the infrastructure software to the applications themselves. Snap furthermore provides flexible telemetry data processing capabilities and a wide range of storage backends. Finally, MIKELANGELO uses two commonly used deployment targets: Cloud and HPC. The purpose of integrating the aforementioned components into a common environment is to demonstrate the potential of the individual components. Cloud integration presents the architectural design and gives details on how the integration with OpenStack is achieved. There is the updated CI integration component Scotty and also a new component dedicated to live re-scheduling of resources in Clouds, called Mikangelo Cloud Manager (MCM). Another tool introduced is called OSmod and is utilized to modify the host operating system. Integration of all components eligible for HPC environments is achieved by extending the widely-spread batch system management software Torque, under the name of vTorque. It extends PBS/Torque by virtualization capabilities to handle VM instantiation, job deployment and VM tear down within the Torque job life cycle in a transparent way to the user.

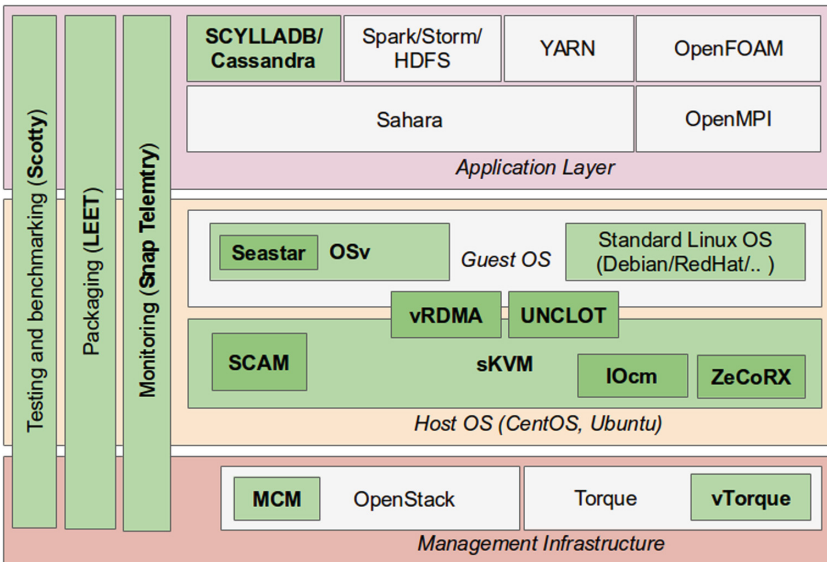


Fig. 1. High-level Cloud and HPC-Cloud architecture.

3 Evaluation and Validation

There are four use cases to validate the project results which provide a perspective to transition project results into active exploitation. The use cases are: big data, aerodynamic maps, cancellous bones and a cloud bursting use case. These cover the most important areas of applications for Cloud and HPC. Cloud computing is tackled by the big data use case, by the aerodynamics use case, and by the cloud bursting use case. Big data is specifically handled by the big data use case. It deploys a virtualized big data platform in the cloud with an automated deployment of synthetic workloads and defined real-world workloads for validation. The cloud bursting use case has developed a new IO scheduler for ScyllaDB, an IO tuning component for ScyllaDB, it improved a RPC framework, and it enhanced the efficiency of data streaming for database replication. Typical HPC applications are covered by the aerodynamic maps use case and the cancellous bones use case. The aerodynamics use case has created a simulation platform with a dashboard supporting the submission of experiments, defined test cases, implemented application packaging. The cancellous bones use case ported its simulation to OSv. The figures below provide an evaluation of accomplished improvements, beneficial to all four use-case, tackling the most crucial part of today’s virtualization, the I/O (Figs. 2 and 3).

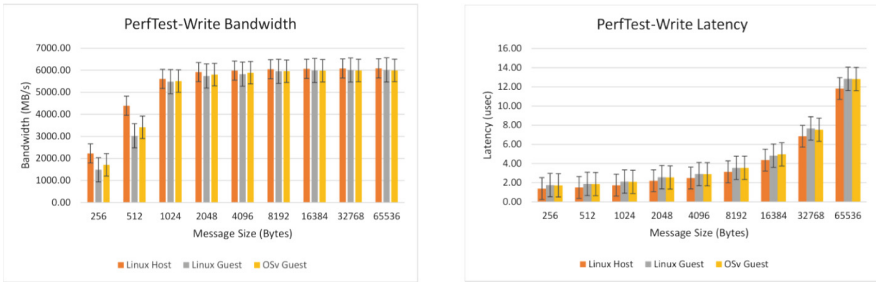


Fig. 2. vRDMA result

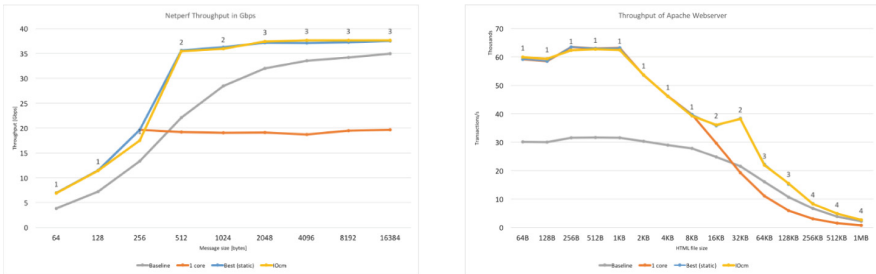


Fig. 3. sKVM I/O


Acknowledgments. The work described in this document has been conducted within the Research & Innovation action MIKELANGELO (project no. 645402), started in Jan 2015, and co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the H2020 framework programme (H2020-ICT-07-2014: Advanced Cloud Infrastructures and Services).

References

1. Drăgan, I., Fortiș, T.-F., Iuhasz, G., Neagul, M., Petcu, D.: Applying self-* principles in heterogeneous cloud environments. In: Antonopoulos, N., Gillam, L. (eds.) *Cloud Computing*. CCN, pp. 255–274. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54645-2_10
2. Felter, W., Ferreira, A., et al.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171–172, March 2015
3. HLRS: Inside (2015). <http://inside.hlrs.de/editions/15spring.html#mike>
4. Adaptive Computing Enterprises, Inc.: Torque resource manager (2017). <http://www.adaptivecomputing.com/products/open-source/torque/>
5. Cloudivus Systems: OSv (2017). <http://osv.io/>



BASMATI: Cloud Brokerage Across Borders for Mobile Users and Applications

Emanuele Carlini^{1,7}(✉) , Massimo Coppola^{1,7}, Patrizio Dazzi^{1,7},
Konstantinos Tserpes^{2,7}, John Violos^{2,7}, Young-Woo Jung^{3,7},
Ganis Zulfa Santoso^{3,7}, Jorn Altmann^{4,7}, Jamie Marshall^{5,7}, Eric Pages^{6,7},
and Myoungjin Kim^{6,7}

¹ ISTI-CNR, Pisa, Italy

`emanuele.carlini@isti.cnr.it`

² NTUA, Athens, Greece

³ ETRI, Daejeon, South Korea

⁴ SNU, Seoul, South Korea

⁵ Amenesik, St Pierre Les Nemours, France

⁶ ATOS, Barcelona, Spain

⁷ INNOGRID, Seoul, South Korea

Abstract. BASMATI aims at delivering an integrated platform that will support the dynamic needs of mobile applications and users through an end-to-end approach for cloud services. BASMATI will emphasize on enabling runtime adaptation of all assets, including user and application prediction models, federation patterns, resources and data management policies, brokerage and offloading decisions. BASMATI platform will coordinate all assets to react in response to real-world events in real-time.

1 Introduction

The explosion in numbers of mobile applications that we are using in our daily lives is unquestionable. Almost all of them are relying on some sort of web-based backend, usually in the form of a restful API or similar. The backend services are commonly delivered through cloud computing technologies, exploiting their ability to adapt to the demand and ultimately allowing more cost-effective solutions to the application providers. However the inherent characteristic of mobile apps, that is user mobility, can combine with other application contextual characteristics (e.g. highly asymmetric load bursts) and put a strain on the computing and storage infrastructure, that needs to compensate any QoS loss caused by the network. A standard approach to resolve the problem and avoid violating the SLA objectives is to leverage resources that are close to the users physical location for hosting backend application services, in a way similar to *edge computing*. However, this solution also comes at a cost and implies some tradeoffs. At a local

scale, a high spatial density of the use of mobile services can reduce the overall efficiency or unreasonably increase the cost of resource provisioning through scaling. At a global scale, the continuous maintenance of resource pools around the world has practical issues and drawbacks that can impair the cost-efficiency of such a solution.

BASMATI¹ is a joint South-Korean and EU Horizon 2020 project, active from June 2016 to August 2018 (26 months), that aims to develop an integrated brokerage platform targeting federated clouds that supports the dynamic needs of mobile applications and users. To tackle the issues mentioned above, BASMATI resolves to the development of a platform that will be able to: (a) support mobile app services and their context; (b) manage the infrastructure to achieve cost-effectiveness; (c) supporting federation at a cloud infrastructure level, and; (d) intelligent offloading and brokering of tasks including edge resources.

1.1 BASMATI Objectives

The BASMATI platform has been conceived, designed and developed having in mind the peculiar requirements of the project use cases. However, the aim of the platform, currently still in development, is to address the project objectives in a broad sense. The consortium thus identified several challenges that can be summarized around the following three main pillars.

Users and Application Knowledge. In order to be able to properly react to the dynamic conditions affecting the cloud infrastructure supporting the backend of mobile applications, it is of paramount importance to understand the actual behavior of applications and their users, as well as to predict the future behavior of the two. BASMATI approaches these challenges by profiling the footprint of applications and analyzing the movement of users. This actually happens at two levels, macro and micro; the former focused on the movements of masses of users, the latter on the movement of single users within a defined area. The joint processing of the application- and user-derived information, that we call situational knowledge, gives to the BASMATI platform the ability of taking into account the actual characteristics of users and applications when modelling their behavior.

Resource Management. The second pillar of challenges faced by BASMATI is contextualized in the area of Resource Management. When it is clear how applications and users will behave, it is fundamental to have a platform that can properly react in a consequent manner. To this end, the BASMATI platform needs to offer innovative ways to represent and manage cloud resources, to properly organize such resources in a federation and to provide effective ways to conduct resource brokering and select the best candidates for a given application to host. For these tasks, current candidates approaches are based on meta-heuristics, such as Genetic Algorithms [2].

¹ <http://www.basmati.cloud/>.

Application Adaptation. The third and last pillar on which the BASMATI challenges are organized focuses on the actual adaptation of applications at runtime. Such support is a key enabler in a widely distributed and dynamic environment in which applications go through changes both required to enable their migration, as well as to make them suitable to offer appropriate performances. This is particularly evident when the conditions of the platform changes and when users move around.

1.2 Project Consortium

The BASMATI project brings together a consortium composed by research centres, universities, industrial companies and SMEs with a proven track record in various facets of the wide spectrum of Cloud Computing technologies and strong commitment towards innovation, and therefore well positioned to efficiently and effectively reach the objectives defined in the paper. The partnership involves 9 well-established organisations from Europe (Greece, Italy, Germany, Spain and France) and Korea. It comprises a leading European University (NTUA) with a high reputation in Cloud Computing, one of the largest European R&D and technology transfer performers (CNR), two important European ICT Industries, with a proven expertise in EU research projects (ATOS and CAS); a leading University in Korea (SNU) and the most important research center in Korea, world leader for the number of US patents (ETRI), and two SMEs specialized in Cloud-based solutions (AMEN and INNO) (Table 1).

Table 1. BASMATI consortium

| Organization name | Short name | Country |
|--|------------|-------------|
| Institute of Communications and Computer Systems/National Technical University of Athens (Coordinator) | ICCS/NTUA | Greece |
| Consiglio Nazionale delle Ricerche | CNR | Italy |
| CAS Software AG | CAS | Germany |
| Atos Spain, SA | ATOS | Spain |
| Amenesik | AMEN | France |
| Electronics and Telecommunications Research Institute (Coordinator) | ETRI | South Korea |
| InnoGrid | INNO | South Korea |
| Seoul National University | SNU | South Korea |

2 Results and Current State

During the 1st year, BASMATI put emphasis in defining the offloading and federation scenarios that are going to be supported by the platform. The criteria

for prioritizing such multiple scenarios were based upon application needs, as well as upon available resources within the project lifetime. As such, the project aimed at developing tools that would allow (a) the intelligent use of specialized devices and cloudlets at the edge of the computing and storage infrastructure, bringing about mobile services while being closer to the end user (thus alleviating network constraints) [3], and; (b) smart migration of services and data to cloud resources owned by different providers, under different provision policies and business models [4,5]. The consortium refers to the first case as offloading and to the second one as federation, seeking mainly cost-effective approaches to implement them in the frame of the pilot scenarios.

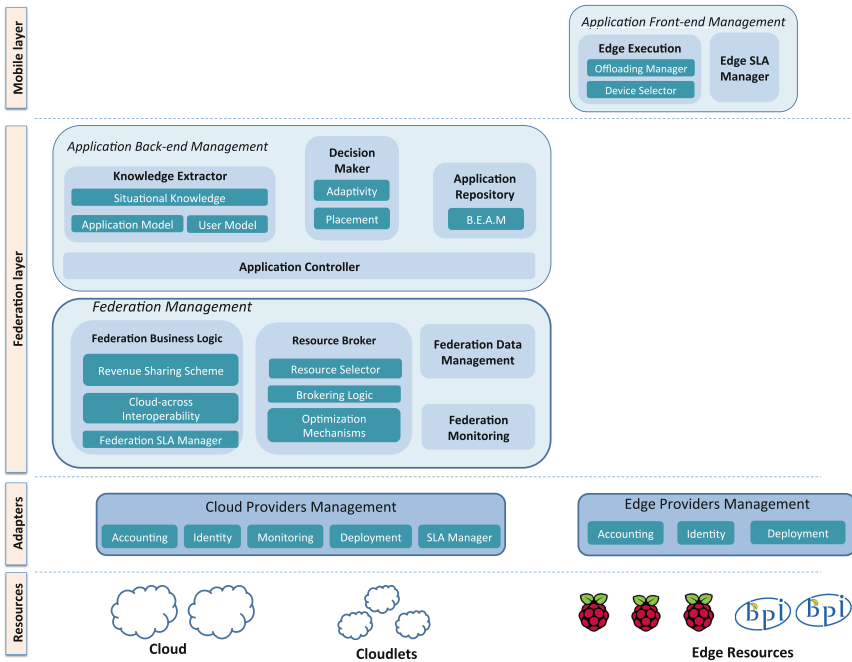


Fig. 1. BASMATI architecture

2.1 Architecture

The idea behind the design choices underlying the architecture is to provide a common ground to address key technological and research challenges that are identified by the requirement analysis, performed also in according to the knowledge and capabilities of the partners and the needs of the project, and which mainly focus on three core aspects [1]: (i) User, application and situation modelling and understanding; (ii) Runtime adaptivity and reconfiguration; (iii) Brokering and Offloading of application and services.

The BASMATI architecture is organized into multiple layers (see Fig. 1). The Mobile layer contains components and functions that are expected to run on the end-user mobile devices, which interface to the lower layers of the BASMATI platform. The Federation Layer represents the core part of the BASMATI infrastructure, and it is further decomposed in the two following logical layers. The Application Back-End Management, which manages the runtime execution of application on top of federated resources, and the Federation Management, which provides the specific features of BASMATI federation by building on top of standard (multi) Cloud features. The Adapter Layer contains those software modules designed to interface BASMATI federation with different Cloud providers. Finally, the Resources Layer groups together actual resources and services from different Cloud providers, Cloudlets and edge resources.

As the development of the component prototypes continues and the overall design is evolved and revised, sequential diagrams for static and dynamic scenarios are analysed. Such diagrams describe the high level interactions of the architecture components, both in a static scenario (i.e. place a new application to the cloud, terminate it) and in a dynamic one (e.g. what happens when an application violates the QoS requested by the users, which and how corrective actions can be performed).

2.2 Cloud Federation Infrastructure

According to the BASMATI terminology, a cloud federation is a result of, primarily, a business agreement between resource providers (not necessarily resource owners like Amazon) who team up in order to make their resources available to other members of the federation at a cost. Thus, the implementation and use of a cloud federation needs to attain a plethora of, often conflicting, objectives. The Cloud Provider Management, the Federated Resource Broker, the Federation Data Management (including Federated SLAs) and, mainly, the Federation Business Logic are the primary components dealing with this goal.

References

1. Altmann, J., Carlini, E., Coppola, M., Dazzi, P., Ferrer, A.J., Haile, N., Jung, Y.W., Kang, D.J., Marshall, I.J., Tserpes, K., et al.: Basmati-a brokerage architecture on federated clouds for mobile applications. Technical report, Seoul National University, Technology Management, Economics, and Policy Program (TEMEP) (2016)
2. Anastasi, G.F., Carlini, E., Coppola, M., Dazzi, P.: QoS-aware genetic cloud brokering. *Future Gener. Comput. Syst.* **75**, 1–13 (2017)
3. Carlini, E., Coppola, M., Dazzi, P., Mordacchini, M., Passarella, A.: Self-optimising decentralised service placement in heterogeneous cloud federation. In: 2016 IEEE 10th International Conference on Self-adaptive and Self-organizing Systems (SASO), pp. 110–119. IEEE (2016)

4. Makris, A., Tserpes, K., Anagnostopoulos, D., Altmann, J.: Load balancing for minimizing the average response time of get operations in distributed key-value stores. In: 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), pp. 263–269. IEEE (2017)
5. Uzbekov, A., Altmann, J.: Enabling business-preference-based scheduling of cloud computing resources. In: Bañares, J.Á., Tserpes, K., Altmann, J. (eds.) GECON 2016. LNCS, vol. 10382, pp. 225–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61920-0_16



C4E: Cloud Brokering Platform for Federated Services Aimed at European Public Administrations

Antonino Galletta¹(✉), Oliver Ardo², Antonio Celesti¹, Peter Kissa²,
and Massimo Villari¹

¹ Dep. Ingegneria, University of Messina, Messina, Italy
{angalletta,acelesti,mvillari}@unime.it

² InterWay a.s., Bratislava, Slovakia
{oliver.ardo,peter.kissa}@interway.sk

Abstract. Cloud computing is evolving towards federated ecosystems, where several Cloud Service Providers (CSPs) federate each other in order to achieve economy of scale and an efficient use of their resources. In line with the Digital Single Market (DSM) strategy promoted by the European Commission, the objective of the FP7 Cloud for Europe (C4E) project is to leverage the concept of Cloud federation in order to pave the way toward an improvement of European Public Administration (PA) ICT services. In particular, it aims, on one hand, at preserving national authorities and on the other hand, at promoting efficiency, collaboration and harmonization among the different PA offices spread over the European Union (EU).

Keywords: Data export control · Cloud Brokering
Federated services · Hybrid storage · C4E

1 Introduction

In line with the Digital Single Market (DSM) strategy promoted by the European Commission, Cloud for Europe (C4E, <http://www.cloudforeurope.eu/>) is a FP7 project co-funded by the European Commission under the funding scheme of “CP-CSA” for PCP. The goal of C4E is to support the European Cloud Partnership allowing the Public Administrations (PAs) to implement strategies based on Cloud federation in order to remove the obstacles of the Cloud usage and to standardize requirements from different European countries.

The consortium of the project is composed of 24 partners coming from 12 countries. Leader of the project is Fraunhofer-Institute for Open Communication Systems (FOKUS), Germany. The Pre-Commercial Procurement (PCP) founding schema is managed by AGID (Italy) that announced a public tender named **Realization of a research and development project PCP (Pre-Commercial Procurement) on “Cloud For Europe”**. The tender was divided into three lots:

- LOT 1: “Federated Certified Service Brokerage (FCSB)”- CIG: 6027774476
- LOT 2: “Secure, Legislation - Aware Storage (SLAS)” - CIG: 6027802B8F
- LOT 3: “Legislation Execution (LE)”- CIG: 602781022C

In this paper we specifically focus on the first two LOTS. The rest of this paper is organized as follows. Section 2 presents our consortium, Section 3 shows the C4E tender objectives, whereas in Section 4, we discuss preliminary results.

2 Consortium

The consortium of LOTS 1 and 2 was composed in order to cover both research and innovation aspects. It includes an academic partner and three industrial partners belonging to different European Countries.

1. **University of Messina (ITA)** (academic), <http://www.unime.it/en/home>
2. **Interway s.r.o. (SVK)** (industrial), leader of LOT 1, <https://www.interway.sk/>
3. **Liberologico s.r.l. (ITA)** (industrial), <http://www.liberologico.com/>
4. **TeamNet International SA (ROU)** (industrial), leader of LOT 2, <http://www.teamnet.ro/>

3 C4E Tender Objectives

C4E [1], is aimed at creating new ICT solutions for PAs fostering the DSM. In particular, the C4E project aims at overcoming the challenges of the adoption of Cloud federation in the PA sector. Such challenges are defined by a technical specification for *Federated Certified Service Brokerage of EU Public Administration Cloud* and comes from the analysis of position and perception of the European public sector about Cloud Computing and the vendor’s Cloud services offering. A C4E reference scenario take into account various PA offices, Cloud Service Providers (CSPs), and citizen where Legal aspects related to ICT services have to be considered. In particular, the C4E tender defines the main rules regarding the use of Cloud services by Public Administrations. The brokering platform described above is the first of the three identified lots (FCSB). LOT 2 (SLAS) regards secure storage in terms of:

- strong and secure encryption of data stored outside the public administration’s legislation;
- long-time storage availability for several type of data;
- compliance with legal requirements in terms of data privacy depending on the type of data.

An important requirements of C4E is the integration among deployed solutions developed into specific LOTS.

4 C4E Tender Results

The project started in June 2013 and ended in March 2017. The final event was in Vienna hosted by the Federal Computing Centre (BRZ) on June 21st, 2017.

Here, we present the achieved results for both LOTs 1 and 2. Regarding the FCSB LOT, the outcome of this project is the accomplishment of an innovative architecture able to deal with Cloud Brokering aspects for allowing European PAs to rely on cross-border services, accessed in federated manner. In particular, the piece of framework we propose is able to deal with more federated Clouds [2] taking into account more aspects in the context of Cloud Brokering where Geographic Constraints are considered. The flow chart in Fig. 1 describes how the selection and the screening of services was performed. Looking at Fig. 1, it is possible to notice that the recommendation engine relies on two main parts: the NoSQL DB part dealing with Basic Rules, and BigData Analytics Part dealing with Advanced Rules linked to Social, Reputation and other Services which are useful for driving the PA officers in making their decisions. The early part is MongoDB based whereas the latter, that is the BigData Engine works using Apache Spark. MongoDB represents the unstructured long-term storage able to be sharded among more sites along with the capabilities to be queried in scalable ways like using geo-location properties. Geo-location features are very effective in our system, because they allow users to define constrains based on geographic position of services and resources. Our system, contrarily to traditional approaches, is able to guarantee continuity of service even if services and resources change their properties. In fact, we only have to update the area of interest in Europe, driven by any kind of needs, changing a document into a specific MongoDB collection and the system is up and ready for the new configuration on fly.

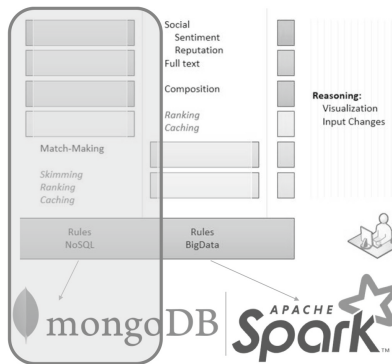


Fig. 1. Service recommender; the recommendation system engine relies on two main parts: NoSQL MongoDB and Apache spark.

Regarding the SLAS, a Cloud application able to manage tagged documentations and media files through Private and Multiple Public Cloud platforms

was created. Specifically, the last one exploits the fragmentation process of the content to store small parts of it into different Cloud platforms. As a result, CSPs can not rebuild the data by its self, ensuring a good level of both data availability and privacy.

The Web Application is composed by two sides: front-end (FE) and back-end (BE). The FE was the result of a MEAN (MongoDB, Express, AngularJS, Node.js) Stack application, developed using Meteor, an open source platform for web, mobile, and desktop. Users are able to insert and download tagged documentations and media files, selecting one option among multiple public and private Clouds. Whereas, the BE was composed of a MongoDB instance, for storing users' files and tags, and several REST server APIs, developed in Java, that act as interface among FE, Cloud Storage Providers and MongoDB storage engine.

Figure 2 shows as healthcare documentations and media files, such as, e.g., Digital Imaging and Communications in Medicine (DICOM), are saved and downloaded by users through a web application, keeping the level of privacy required. We remark that the deployed system works with all type of files. Figure 2 shows the overall architecture. Specifically, the system was composed of the following elements:

1. physicians, the users of our system;
2. Magnetic Resonance Machine (MRM) that produces MRIs;
3. meteor web-app, an app that implement the recomposition and allows users to analyze DICOM files directly from their browser;
4. OwnCloud, that acts as Private Cloud;
5. several Public Clouds, used to share content among physicians;
6. Splitter, a module that splits and disseminates data files;

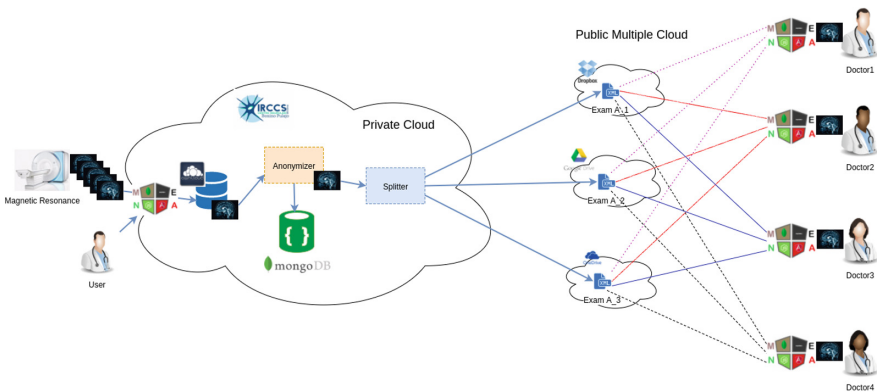


Fig. 2. Overall high level federation architecture

7. MongoDB, a NoSQL database used to contain patients personal data and TAGs; and
8. Anonymizer, a module that obfuscates patients' data.

Producer and Consumers generated and retrieved MRI files codified as DICOM. In particular, Producer was represented by the MRM and makes clinical tests, which generates lots of images stored inside specific private Cloud folders. In these directories, few internal physicians gain the access, because the private Cloud guarantees data confidentiality and privacy. On the other hand, Consumers can be considered as internal or remote physician, which analyze the DICOM files thorough the Web application. In order to increase the overall security of the system, DICOM files were processed by an ad-hoc module that anonymize them. The algorithm modifies the DICOM header and stores original data into specific MongoDB collections. Thus, external physicians makes diagnosis without be able to associate the series with patients. Moreover, the Splitter module receives the anonymized DICOM series as input and divides it into chunks for distributing them among multiple public Cloud Storage providers selected through the recommender deployed in FCSB LOT.

Acknowledgment. This work has been supported by Cloud for Europe (C4E) Tender: *Realization of a research and development project PCP (Pre-Commercial Procurement) on “Cloud For Europe”*, Italy-Rome: Research and development services and related consultancy services Contract notice: 2014/S 241-424518. Directive: 2004/18/EC. (<http://www.cloudforeurope.eu/>)

References

1. Agenda Digitale Italia (AGID): Technical offer template. Retrieved July 2017 http://www.agid.gov.it/sites/default/files/documentazione/technical_offer_template_03-12-14_15.30_publish_0_0.pdf/
2. Celesti, A., Levin, A., Massonet, P., Schour, L., Villari, M.: Federated networking services in multiple openstack clouds. In: Celesti, A., Leitner, P. (eds.) ESOC Workshops 2015. CCIS, vol. 567, pp. 338–352. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_26

Author Index

- Allam, Salma 73
Altmann, Jorn 181
Anderson, Maya 154
Andrikopoulos, Vasilios 57
Ardo, Oliver 187
Ayed, Dhouha 149
- Bekri, Moulay Ali 73
Berginc, Gregor 175
Bermbach, David 154
Boyle, John 149
Brogi, Antonio 130
- Cappiello, Cinzia 154
Carlini, Emanuele 181
Carnevale, Lorenzo 73
Casale, Giuliano 164
Celesti, Antonio 187
Chen, Fang 175
Chesta, Cristina 159
Chronz, Peter 175
Činkelj, Justin 175
Coppola, Massimo 181
- Damiani, Ferruccio 159
Dazzi, Patrizio 181
de Laat, Cees 169
- El Ouahbi, Rachid 73
Elango, Divyaa Manimaran 86, 117
- Fan, Shiqing 175
Fowley, Frank 86, 117
- Galletta, Antonino 73, 187
Garcia-Perez, David 154
Gilboa, Niv 175
- Har'El, Nadav 175
Heisel, Maritta 149
Hinkelmann, Knut 35
- Johnsen, Einar Broch 159
Jones, Andrew 169
Joosen, Wouter 5
Jung, Young-Woo 181
- Kat, Ronen I. 154
Kennedy, John 175
Kim, Myoungjin 181
Kissa, Peter 187
Kritikos, Kyriakos 20, 35
- Laurenzi, Emanuele 35
Li, Chen 164
López, Manuel Ramírez 102
- Makki, Majid 5
Mann, Zoltán Ádám 149
Marinakias, Achilleas 154
Marshall, Jamie 181
Martin, Paul 169
Meth, Kalman 175
Metzger, Andreas 149
Moulos, Vrettos 154
Mundt, Paul 149
- Pages, Eric 181
Pahl, Claus 86, 117
Pallas, Frank 154
Paravoliasis, Andreas 20
Pernici, Barbara 154
Plebani, Pierluigi 154
Plexousakis, Dimitris 20
- Rinaldi, Luca 130
Røst, Thomas Brox 159
- Salado, Guadalupe Flores 169
Salant, Eliot 149
Sandoval, Yosandra 175
Santoso, Ganis Zulfa 181
Scalosub, Gabriel 175

Seidl, Christoph 159
Soldani, Jacopo 130
Spillner, Josef 102
Stankovski, Vlado 169
Struckmann, Nico 175
Suciu, George 169
SurrIDGE, Mike 149

Tai, Stefan 154
Taylor, Ian 169
Toffetti, Giovanni 102
Tserpes, Konstantinos 181

Ulisses, Alexandre 169

Van Landuyt, Dimitri 5
Villari, Massimo 73, 187
Violos, John 181
Vitali, Monica 154

Yu, Ingrid Chieh 159

Zeginis, Chrysostomos 20
Zhao, Zhiming 169