Handling Cloud PaaS Platforms Heterogeneity

by

Eman Hossny Abdelghony

A thesis submitted in partial fulfillment for the degree of

Doctor of Philosophy in Computer Science

under supervision of

Prof. Fatma A. Omara

Prof. Hesham A. Hassan

Dr. Sherif A. Khatab

in the
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF COMPUTERS AND INFORMATION
CAIRO UNIVERSITY
EGYPT

April 2017

Declaration of Authorship

I, Eman Hossny, declare that this thesis titled, 'Handling Cloud PaaS Platforms Heterogeneity' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

Acknowledgements

At first, I thank Allah who provided me with all the countless success reasons and the fruitful environment by which this work is fulfilled. I am also so grateful to my supervisors, Dr. Sherif Khatab, Prof. Fatma Omara, and Prof. Hesham Hassan, for providing continuous support, guidance, mentoring, and technical advice over this thesis. In addition to their continuous review of the thesis and its related research articles. Finally, I thank my family for their incredible patience and support throughout the lifetime of this thesis.

Part of this work was done in the ITEA2 EASI-CLOUDS project (http://easi-clouds.eu). The Egyptian consortium in the project was funded by ITIDA ITACfrom 10/2012 to 1/2014.

Abstract

In Platform as a Service (PaaS) cloud model, the providers have their own specific-APIs (i.e., heterogeneous). These specific-APIs make developers to be locked inside a specific platform and not able to easily port their applications among different platforms. As a result, vendor lock-in problem appeared. One solution to this problem is to use generic-APIs with specific adapters to implement portable applications. However, any update in a PaaS specific-API makes its corresponding adapter unusable which causes what is called API dynamic adaptation problem. Therefore, this thesis provides two main contributions; proposes a framework, called STAGER, to overcome the API dynamic adaptation problem and proposes generic-APIs to overcome the PaaS vendor lock-in problem. STAGER is a framework for semantic-based generation of generic-API adapters for portable cloud applications. STAGER aims to generate the adapters of any semantically annotated generic-APIs. Furthermore, this thesis proposes service-based generic-APIs, called Std-PaaS APIs, that can be used by cloud developers to implement generic applications. These Std-PaaS APIs can include a set of generic-APIs for each PaaS service (e.g., blob storage, datastore, and messaging). Currently, the Std-PaaS APIs include two generic-APIs for blob storage and NoSQL datastore services. In addition, this thesis extends the COAPS generic deployment API to include deploying applications on GAE platform, besides CloudFoundry (CF) and OpenShift (OS). Thus, the COAPS API helps cloud developers to deploy their applications on heterogeneous PaaS platforms. STAGER was evaluated using the Std-PaaS APIs by generating their adapters for two PaaS platforms; Google App Engine (GAE) and Windows Azure. Although there is some overhead for semantically annotating the PaaS APIs, the evaluation results prove the feasibility of STAGER framework and promote the usage of the generated adapters for implementing portable cloud applications. Moreover, in the state of the art approaches, the code generation is based on one-to-one mapping between the generic-APIs and its corresponding specific ones, whereas our code generation approach is based on SPARQL queries which is more flexible and it can map each generic method into one or more specific methods.

Contents

List of Figures

List of Tables

Abbreviations

ACCORDS	**A**dvanced **C**apabilities for **C**ompatible**O**ne **R**esource **D**escription **S**ystem
ACL	**A**ccess **C**ontrol **L**ist
API	**A**pplication **P**rogramming **I**nterface
BLOB	**B**inary **L**arge **OB**jects
CORDS	**C**ompatible**O**ne **R**esource **D**escription **S**ystem
CAPEX	**CAP**ital **EX**penses
CBSS	**C**loud **B**lob **S**torage **S**ervice
CF	**C**loud**F**oundry
COAPS	**C**ompatible **O**ne **A**pplication and **P**latform **S**ervice
CDMI	**C**loud **D**ata **M**anagement **I**nterface
DAML	**D**ARPA **A**gent **M**arkup **L**anguage
IaaS	**I**nfrastructure **a**s **a** **S**ervice
IDE	**I**ntegrated **D**evelopment **E**nvironment
GAE	**G**oogle **A**pp **E**ngine
GCD	**G**oogle **C**loud **D**atastore
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NoSQL	**N**ot **o**nly **S**tructured **Q**uery **L**anguage
OASIS	**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards
OCCI	**O**pen **C**loud **C**omputing **I**nterface
OPEX	**OP**erating **EX**penses
OS	**O**pen**S**hift
OWL	**W**eb **O**ntology **L**anguage
PaaS	**P**latform **a**s **a** **S**ervice
PADM	**P**aaS **A**pplication **D**escription **M**odel
PBS	**P**latform **B**asic **S**ervices

RDF	**R**esource **D**escription **F**ramework
RDFS	**R**esource **D**escription **F**ramework **S**chema
SaaS	**S**oftware **a**s **a** **S**ervice
SAVG	**S**emi-automatic **A**dapter **V**alidation **G**eneration
SCOP	**S**ource **C**ode **O**ntology **P**opulation
SQL	**S**tructured **Q**uery **L**anguage
SPARQL	**S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
STAGER	**S**eman**T**ic-based Gen**ER**ation of **G**eneric-API **A**dapters
Std-PaaS APIs	**St**andar**d** **P**latform **a**s **a** **S**ervice **A**pplication **P**rogramming **I**nterface
VM	**V**irtual **M**achine
WAR	**W**eb **A**pplication **A**rchive

Chapter 1

Literature Review

As it is stated previously in Chapter **??**, the vendor lock-in problem is considered as the second important challenge which prevents cloud adoption. Therefore, many efforts are done to overcome this problem and support cloud portability. Petcu and Vasilakos [1] have categorized these efforts into four types based on the methodology that is used to solve the vendor lock-in problem; model driven engineering (MDE), standards, open/-generic APIs, and semantics. In this thesis, we propose an integrated framework, called STAGER, that uses two methodologies, semantics and generic APIs, to overcome the PaaS vendor lock-in problem and support application portability. The proposed generic APIs aims to hide the heterogeneity of different cloud services and help cloud developers to implement cloud applications that are agnostic to PaaS platforms. Moreover, the proposed generic APIs provide a set of specific adapters to allow the mapping between the generic APIs and the target services. On the other hand, the semantics are used to semi-automatically generate the specific adapters.

However, Gonidis et al. [2] have identified four main characteristics that may hamper cloud portability as follows:

- **Programming languages and frameworks**; each PaaS platform provides a set of heterogeneous programming languages and frameworks.
- **Data storage**; each PaaS platform provides a set of heterogeneous types of data storage, which can include blob storage, relational databases, NoSQL datastores, and message queues.

- **Platform specific services**; each PaaS platform provides a set of specific services, such as sms, email, and monitoring services, with specific APIs. Cloud developers can use these services through their specific APIs.

- **Platform specific configuration files**; some PaaS platforms requires a specific configuration file, which includes information about how to deploy and execute the application. For example, GAE requires *appengine-web.xml* file, which includes the application ID that will be used in the deployment process.

In addition, da Silva et al. [3] have categorized the cloud portability into four types; virtual machine (VM) portability, IaaS application portability, PaaS application portability, and data portability. This research focuses on applications portability over PaaS clouds. So, we will classify the current related work into three categories; standardization efforts, research projects, and generic APIs. Section 1.1 elaborates the current standardization efforts to overcome the vendor lock-in problem. In addition, the current research projects to overcome this problem is presented in Section 1.2. Section 1.3 elaborates the currently available generic APIs to solve the lock-in problem. Finally, a brief discussion is introduced in Section 1.4.

1.1 Standardization Efforts

In this Section, we will explore a set of standardization efforts that has been done to address the cloud vendor lock-in problem, such as Cloud Application Management for Platforms (CAMP), Topology and Orchestration Specification for Cloud Applications (TOSCA), Open Cloud Computing Interface (OCCI), Cloud Data Management Interface (CDMI), and Open Virtualization Format (OVF).

CAMP; OASIS has introduced Cloud Application Management for Platforms (CAMP) standard, which provides standard management API to allow **deploying** and **managing** (i.e., starting, stopping, monitoring, billing, etc.) applications over different PaaS platforms [4]. CAMP API is a RESTful API that is based on JSON. Therefore, it is independent on PaaS platforms and programming languages. Each application has a set of requirements and each PaaS platform has a set of capabilities. So, CAMP aims to match the application's requirements against the PaaS's capabilities and generate

deployment plans. Meanwhile, CAMP represents the deployed application as *assembly template*. Finally, CAMP combines the assemblies and the deployment plans into Platform Deployment Package (PDP), which can be used to migrate an application from a PaaS platform to another. Therefore, CAMP has two main advantages:

- It can be implemented as a plug-in inside the application development environment.
- It helps cloud developers to port their applications among different PaaS platforms using a standard management API.

However, if the ported application is implemented using a PaaS proprietary API (e.g., GAE datastore), then it cannot be ported to another PaaS platform that uses another proprietary API (e.g., Azure table storage). Furthermore, PDP deployment is only supported by a very small number of new PaaS platforms, such as Solum[1] and Brooklyn[2].

TOSCA; OASIS has provided Topology and Orchestration Specification for Cloud Applications (TOSCA) standard, which provides an XML-based modeling language to automatically **deploy** and **manage** cloud applications [5]. In TOSCA, cloud applications can be specified by a topology and a set of management plans. A topology is used to describe the application's components and their relationships, whereas management plans describe the management operations that need to be executed to deploy and manage a cloud application. The management plans are defined using standard workflow languages, such as BPEL[3] and BPMN[4]. Finally, TOSCA combines the application's topology and management plans into a Cloud Service Archive (CSAR) that is ready to be deployed on a PaaS platform. TOSCA standard is better than CAMP standard because it supports the re-usability of the application's components. However, most of the currently available PaaS platforms could not support the CSAR deployment except a very small number of PaaS platforms, such as OpenTOSCA[5] ecosystem.

OCCI; Open Grid Forum (OGF) has introduced Open Cloud Computing Interface (OCCI) standard, which provides standard RESTful APIs for **managing** clouds [6].

[1] https://wiki.openstack.org/wiki/Solum
[2] https://wiki.apache.org/incubator/BrooklynProposal
[3] Business Process Execution Language
[4] Business Process Model and Notation
[5] http://www.opentosca.org/

OCCI is started by creating a remote management API for IaaS model then it is evolved to manage the other two models (PaaS and SaaS). There are many IaaS platforms that are based on OCCI standard, such as jclouds, OpenStack, BigGrid, and OpenNebula. Moreover, COAPS API, which is a generic PaaS API for deploying and managing cloud applications on multiple heterogeneous PaaS platforms, is based on OCCI standard [7]. Currently, the COAPS API supports two PaaS platforms; Cloud Foundry (CF) and Openshift (OS). In a previous work, we have implemented a new adapter in the COAPS API to allow deploying cloud applications on GAE, besides CF and OS [8, 9]. The COAPS API is similar to IBM Altocumulus project [10] which provides users with a uniform interface to allow them to deploy their applications on multiple PaaS platforms such as IBM HiPODS, Amazon EC2, and GAE.

CDMI; SNIA organization has provided Cloud Data Management Interface (CDMI) as an international standard to manage cloud storage [11]. CDMI is similar to Amzaon S3 (Simple Storage Service). However, Amazon S3 is a proprietary solution. CDMI provides a standard interface to help cloud developers to execute CRUD (i.e., create, read, update, and remove) operations on data elements of IaaS cloud storage. Moreover, CDMI provides another standard interface to manage IaaS cloud storage. Examples for these management operations include backup, access control list, manage containers, billing, etc. By this way, CDMI includes two interfaces; *data path* (which is used to store and retrieve data) and *control path* (which is used to manage data). In addition, these interfaces are based on REST technology and they provide a set of standard methods for three data elements; blobs, containers, and message queues. However, CDMI did not support table storage and SQL databases. Livenson and Laure [12] have implemented an open source prototype that is based on CDMI. Their prototype, called CDMI-proxy, provides generic APIs with specific adapters to access two heterogeneous cloud storage; blobs and message queues. However, their specific adapters are implemented manually.

OVF; DMTF organization has proposed an open standard for virtual machines called Open Virtualization Format[6] (OVF). OVF provides a standard format for the heterogeneous virtual machines (VMs) in order to allow porting them among heterogeneous IaaS platforms in an easy way. On the other hand, porting applications over different IaaS clouds can be done by provisioning a VM on the target IaaS cloud with the target

[6]http://www.dmtf.org/standards/ovf

framework (i.e., language and packages), then deploy applications on this VM. In addition, it is possible to build the application inside a container (e.g., Docker[7]) and then migrating the container, which contains the application with its dependencies, to other IaaS clouds [13].

1.2 Research Projects

This Section introduces a set of currently available research projects that has been done to address the cloud vendor lock-in problem, such as Cloud4SOA, PaaSport, mOSAIC, and MODAClouds.

Cloud4SOA; Zeginis et al. [14] have presented the Cloud4SOA project, which aims to provide a harmonized API for **managing** (i.e., deploying, monitoring, migrating) applications across multiple PaaS platforms. Their API is based on semantic and it has a set of specific adapters for some PaaS platforms, such as AWS Beanstalk, Heroku, OS, and CF. However, Cloud4SOA has some overhead because its structure is composed of multiple layers. Furthermore, it requires that the applications are implemented based on SOA.

PaaSport; Bassiliades et al. [15] provides semantically-based cloud-broker which aims to **deploy**, migrate, and **manage** cloud applications over heterogeneous PaaS platforms. In addition, it provides a unified API with specific adapters to support PaaS portability. Their unified API is based on the *CAMP* standard and *Cloud4SOA*. However, their adapters are manually implemented which will lead to API synchronization problem. The PaaSport is similar to the proposed STAGER framework in that the PaaSport requires the PaaS providers to semantically annotate their PaaS platforms through an OWL ontology. Moreover, the PaaSport provides an ontology-based recommendation algorithm which aims to select the most suitable PaaS platform based on the application's requirements, which may be functional or non-functional requirements.

mOSAIC; Petcu et al. [16] have provided the mOSAIC (Open Source API and platform for Multiple Clouds) project, which aims to support applications portability and interoperability among multiple clouds. mOSAIC provides cloud developers with an

[7]https://www.docker.com

open source cloud API with a set of adapters. This API is a generic-API, which aims to **deploy** applications across different IaaS clouds. However, the main drawback of their API is that it was designed based on event-driven, which is a complex programming style. In addition, their API focuses on deploying applications across IaaS clouds. Currently, mOSAIC proposes adapters for a set of IaaS clouds, such as Amazon EC2, OpenNebula, and Eucalyptus. For more info about the mOSAIC project see [17].

As a part of mOSAIC project, Cretella and Martino [18] have presented an approach to semantically annotate cloud APIs in order to allow application portability. The application portability can be satisfied by understanding cloud APIs's functions and then mapping these functions to its corresponding APIs of another provider. They have proposed an approach to semantically analyze and annotate cloud APIs in order to understand their functions. Similar to the STAGER framework, they have used the reflection mechanism to analyze the different cloud APIs and convert them into an ontology. Furthermore, they have proposed an automatic API alignment technique in order to map a given specific API to a given generic API, which called neutral API. However, the output of this alignment technique must be manually validated. In addition, they did not provide a case study to validate their proposed solutions.

Furthermore, Cretella and Di Martino [19] have proposed a semantic engine, as a component of mOSAIC project, to support cloud application portability. Their semantic engine helps cloud developers to semantically detect the required cloud APIs and resources that are suitable to develop their cloud applications. It provides cloud developers with a list of functions related to cloud and a list of patterns related to application design to help them in designing their applications with the required functions. However, this semantic engine is just a prototype and the semantic annotation of cloud APIs has been done statically. Similar to the STAGER framework, they have used the Jena library and Java programming language to implement their semantic engine.

Moreover, Di Martino et al. [20] have proposed a semantic-based common model that can be used to represent cloud applications through design patterns. Their common model aims to support cloud portability and interoperability. They have extended an ontology language for design patterns, called ODOL, to include semantic annotation using OWL-S ontology. Cloud developers can use this extended ontology to design their cloud applications independently of any cloud platform. In order to support application

portability, they have proposed an automatic mapping methodology to transform a designed cloud pattern of an application into its specific implementation of a target cloud platform.

MODAClouds; most of the current research efforts have solved the cloud vendor lock-in problem by providing standards or generic APIs. However, MODAClouds followed a different approach to solve this problem, using model driven engineering (MDE). The main objective of the MDE is to provide *abstraction* and *automation* [2]. The abstraction is satisfied by making the application development is independent from the target PaaS platform, whereas the automation is satisfied by automatically transforming an abstracted application into a specific application to be deployed on the target PaaS platform.

MODAClouds [21, 22] aims to design and execute cloud applications over heterogeneous clouds (IaaS, PaaS, or SaaS). It provides a modeling language, called MODACloudML, to help developers to describe cloud applications. Moreover, it provides an IDE which helps the developers to design their applications independent on any PaaS platform. Then this IDE can semi-automatically map the designed application into code to be ready for execution.

Gonidis et al. [23, 24] provide a MDE framework to hide the heterogeneity of the PaaS specific-APIs. Their framework helps cloud developers to design agnostic cloud applications that can be deployed on heterogeneous PaaS platforms. Their framework is based on semantic ontology and meta-model. It provides an abstract model and generic API (called reference API) for each PaaS service. In addition, it automatically generates the specific adapters of the reference APIs. Their work is similar to our work, however, it differs in three points. Firstly, their framework is based on MDE, while our framework is based on generic-APIs with semantic annotations. Secondly, their framework can only handle the common features of a PaaS specific service among heterogeneous PaaS platforms, while our framework can handle the unique and common features (e.g., the NoSQL datastore requires *partition key* in case of Azure, while it is not required in case of GAE. So, the *partition key* is considered a unique feature that is handled in the specific adapters of our datastore generic-APIs. However, it cannot be handled by the framework of these authors). Thirdly, their code generation approach is based on one-to-one mapping between the generic-APIs and its corresponding specific ones, whereas

our code generation approach is based on SPARQL queries which is more flexible and it can map each generic method into one or more specific methods.

da Silva et al. [3] have proposed a MDE approach to solve the PaaS application portability problem. They have created a meta model to represent the cloud platform's concepts in higher level of abstraction. Based on this meta model, they have proposed a domain specific language (DSL) which helps cloud developers to specify their applications in an abstract way, called Platform Independent Model (PIM). Based on their proposed DSL and PIM, they have defined a set of transformations, called Platform Specific Model (PSM), which can be used to generate the code of the abstracted applications to be deployed on different platforms. They have evaluated their approach on two PaaS platform, GAE and Azure.

Ranabahu et al. [25] have proposed an approach based on abstraction driven in order to support cloud application portability. Their approach is very similar to the MDE, where it helps cloud developers to specify their applications through an abstract language (i.e., DSL). Then, their abstracted applications can be converted, through a code generation engine, into specific applications to be deployed on target PaaS platforms. However, to deploy an abstracted application on a set of PaaS platforms, then it is required to generate a specific application for each one of the target PaaS platforms, which may be time consuming. Moreover, the abstraction cannot present all specific features that are existing in the target PaaS platform. By this way, a generated application need to be manually customized to support specialized features.

1.3 generic APIs

In this Section, we will explore a set of currently available generic APIs that aims to overcome the cloud vendor lock-in problem and help cloud developers to implement cloud applications independently of specific PaaS platforms. These generic APIs will be introduced in chronological order as follows.

Maximilien et al. [26] have proposed a prototype of a middleware that is independent on cloud platforms in order to overcome the cloud vendor lock-in problem. Their middleware provides REST-based generic APIs with specific adapters. These generic APIs

provides methods to execute the CRUD operations of cloud resources. They have implemented the adapters of their generic APIs for three cloud platforms; GAE, AWS, and IBM private cloud.

Loutas et al. [27] have proposed a semantic-based architecture, called RASIC, to help developers to **design** and **deploy** their cloud applications over heterogeneous IaaS platforms. Their architecture is based on semantics and SOA. The semantic annotations are used to annotate cloud applications and resources, whereas SOA is used to help developers to design their applications as web services. Their architecture provides common APIs with static adapters for four heterogeneous IaaS resources, namely: network, storage, instance, and image.

Hill and Humphrey [28] have proposed an abstraction layer, called CSAL, to hide the heterogeneity of different cloud storage services. In addition, CSAL helps cloud developers to port their storage-based applications over heterogeneous cloud platforms. Currently, CSAL provides REST common APIs for three cloud storage services; tables, blobs, and queues. The authors have evaluated their work on two cloud platforms; Windows Azure and Amazon AWS (SimpleDB, S3, and SQS). The main advantage of the CSAL is that it provides a unified namespace to access each one of the supported services (e.g., the unified namespace allows accessing a container using its name without caring about its location). However, the adapters of their common APIs is manually implemented. So, whenever the specific storage APIs of the supported cloud platforms are changed, their adapters will be unusable.

Silva et al. [29] have proposed Service Delivery Cloud Platform (SDCP), which provides service-based common APIs in order to support interoperability among different PaaS platforms. In addition, it provides cloud developers with toolkit to help them in **implementing** their applications. On the other hand, cloud developers can use this toolkit to implement a new plug-in (i.e., adapter) for a specific PaaS provider. However, this adapter is implemented manually. So, whenever a PaaS specific-API is changed, its corresponding adapter will be unusable. Currently, SDCP provides common APIs for three services; blob storage, columnar database, and notification. An encryption method is provided by their storage service's common API in order to encrypt data before uploading it to clouds.

Kolb and Wirtz [30] have proposed a common model for different PaaS platforms to solve the application portability problem. In their model, they have defined the structure of a PaaS platform to be composed of three layers, namely: Infrastructure, Platform, and Management. They converted their common model into a taxonomy which specifies a set of common properties among different PaaS platforms. Each specific PaaS provider specifies his profile which contains specific values for the taxonomy's properties. An application can be ported by matching its profile, which specifies an application's requirements, against the profiles of the different PaaS platforms. The matching process will output a list of PaaS platforms that are suitable for deploying that application. The disadvantage of their model is that they ignore the low-level implementation details while solving the application portability problem. In other words, their model only selects one or more of suitable PaaS platforms to deploy an application and still a cloud developer needs to use a specific PaaS API to implement his application. However, the STAGER framework provides cloud developers with a service-based generic-APIs to help them in implementing portable cloud applications.

Cunha et al. [31] have proposed the *PaaS Manager* framework in order to overcome the vendor lock-in problem and support the cloud applications portability. The PaaS Manager provides a common RESTful API for **deploying** and **managing** cloud applications among different PaaS platforms. Their common API provides a set of specific adapters, which have been implemented manually, for the supported PaaS platforms, such as CF, Heroku, and CloudBees.

Rafique et al. [32] and Walraven et al. [33] have proposed a policy-driven middleware, called PaaSHopper, which aims to **develop** and **deploy** cloud applications on hybrid clouds. The PaaSHopper provides two main layers; an abstraction layer to allow interoperability and portability of cloud applications over heterogeneous PaaS platforms and a policy-driven execution layer to allow deployment and execution of cloud applications. It provides a uniform API, which is Java-based, for three services, namely: blob storage, data storage, and asynchronous tasks. Their work is very close to our work. Their uniform API can be semantically annotated and then imported into the STAGER framework to semi-automatically generate the adapters for the supported PaaS platforms. However, their uniform API is not an open source. On the other hand, the adapters of their uniform API have been implemented manually. Therefore, when a

new PaaS platform is added or when an existed PaaS provider changes its API, then a new adapter need to be (re)implemented (i.e., the adapters are suffering from the API synchronization problem). Furthermore, the PaaSHopper is in its early stage because it only provides a prototype.

CDPort (Cloud Data Portability) framework [34, 35] has solved the data portability problem among different cloud databases. They proposed a standard API and a common data model to hide the heterogeneity of different cloud data storage services. They provide an adapter for each one of the supported databases. They have tested their framework by implementing a portable java application for supporting NoSQL databases. One of the limitations of their portable application is that it requires an update in its source code to support a specific adapter (e.g., when they need to work with Amazon SimpleDB, they need to change some part of their application's code to use the *SimpleDBAdapter*). Another problem in the CDPort framework is that the adapters' source codes are implemented manually. Thus, whenever a specific database API has been modified, the adapters cannot be used and their code needs to be updated manually.

Kolb and Röck [36] and Röck and Kolb [37] have proposed a unified interface, called *Nucleus*, for **deploying** and **managing** cloud applications over different PaaS platforms. Their interface provides language and platform independent API, which is based on REST and JSON. In addition, their unified API includes a set of specific adapters for each one of the supported PaaS platforms, such as CF, OS, and Heroku. However, when a PaaS provider changes its API, its corresponding adapter need to be updated manually (i.e., their adapters suffer from the API synchronization problem).

Brogi et al. [38] have proposed *SeaClouds*, which provides common RESTful APIs with a set of specific adapters for **deploying** and **managing** applications over PaaS and IaaS platforms. In SeaClouds, the applications' components are specified through TOSCA standard. In addition, TOSCA and CAMP standards are used to manage applications at runtime. Although SeaClouds is an open source, it is a language dependent because it is based on JAVA.

Yasrab and Gu [39] have proposed *Multi-Cloud PaaS Architecture* (MCPA), which provides a platform and unified API to **manage** and **deploy** service-based cloud applications over heterogeneous PaaS platforms. The MCPA is adopted from *SeaClouds*.

Moreover, it combines three open source tools, namely: Opscode-Chef, mOSAIC, and P-TOSCA in order to support PaaS portability and interoperability. Opscode-Chef is used to manage cloud applications; whereas mOSAIC is used to discover and monitor cloud applications; and P-TOSCA, which is TOSCA extension, is used to orchestrate and describe cloud applications.

Pallavi and Babu [40] have provided generic RESTful APIs, called Open-PaaS-Database API (ODBAPI), to overcome the heterogeneity of different structured and unstructured databases. Their generic APIs provide generic operations to insert, remove, and update records on heterogeneous DBs. In addition, the authors implemented a set of static adapters, called virtual data stores, to map their generic APIs into their corresponding specific one. Moreover, they have provided a manifest in order to discover the heterogeneous databases and allow applications to be automatically deployed. They have evaluated their work on two case studies, MySQL and MongoDB.

Kumar M and Jose [41] have proposed a Generic Cloud Framework (GCF) to help developers to **develop** and **deploy** generic applications over heterogeneous IaaS platforms. The GCF provides a strategy to implement generic applications that are domain specific, such as applications for medical systems, applications for agriculture systems, etc. Their framework includes a set of generic APIs with a set of static adapters, called wrapper. However, their framework is in its early stage; it is just a prototype and it has been evaluated on only one case study for the agriculture domain. Moreover, they did not specify the IaaS services that the framework already hide their heterogeneity.

Application portability over IaaS clouds can be done by implementing the applications using open generic APIs, such as Jcloud, Libcloud, Fog, and Deltacloud. Jclouds[8] provides Java-based generic APIs as an open source for three services; compute, storage, and load balancer. Fog[9] is similar to jclouds but it is based on Ruby. Libcloud[10] provides python-based unified API for compute and DNS services. Deltacloud[11] provides RESTful generic APIs using Ruby language, for compute and storage services, with a set of drivers to manage a set of different IaaS clouds. Thus, these generic APIs are service and language dependent. However, the adapters of all these generic APIs have

[8] http://jclouds.apache.org/
[9] http://fog.io/
[10] https://libcloud.apache.org/
[11] https://deltacloud.apache.org/

been manually implemented. So, they may suffer from API synchronization problem whenever the specific APIs are changed.

1.4 Discussion

Table 1.1 summarizes the current research efforts to overcome the cloud vendor lock-in problem. Each research effort is identified with respect to four features:

- **Standard**; it means that this research effort is proposed by an organization for standards.
- **Generic Management API**; it means that this research effort provides a generic management API that can be used in the deployment and management phases of a cloud application provisioning. These phases are elaborated in **??**.
- **Generic Implementation API**; it means that this research effort provides a generic implementation API that can be used in the implementation phase of a cloud application provisioning.
- **Cloud Model**; it specifies the cloud model that can be handled by this research effort.
- **API synchronization Problem**; it refers if the generic API's adapters, which are provided by this research effort, suffer from the API synchronization problem.

It should be noted that most of the currently related work focus on providing standards or generic-APIs in order to overcome the cloud vendor lock-in problem and support application portability. These generic-APIs may focus on management operations for deploying and managing cloud applications among different cloud platforms (11 researches out of 20). On the other hand, these generic-APIs may focus on service operations for implementing cloud applications independently of cloud platforms (9 researched out of 20). However, the adapters of the currently available generic-APIs are implemented manually. So, they are suffering from the API synchronization problem. In addition, the current standardization efforts (e.g., CAMP and TOSCA) are in its early stage and they are not supported by most of the leading PaaS platforms.

TABLE 1.1: Research Efforts for Vendor Lock-in Problem

Research Effort	Standard	Generic Manag. API	Generic Implem. API	Cloud Model	API Synchronization Problem
CAMP (2014)	✓	✓	✗	PaaS	All Adapters are suffering from **API synchronization problem**
TOSCA (2014)	✓	✓	✗		
COAPS (2013, 2016)	✗	✓	✗		
Cloud4SOA (2013)	✗	✓	✗		
Nucleus (2016)	✗	✓	✗		
PaaS Manager (2014)	✗	✓	✗		
CDPort (2014, 2015)	✗	✗	✓		
PaaSHopper (2014, 2015)	✗	✗	✓		
SDCP (2013)	✗	✗	✓		
ODBAPI (2016)	✗	✗	✓		
CDMI-proxy (2011)	✗	✗	✓		
SeaClouds (2016)	✗	✓	✗	IaaS PaaS	
MCPA (2016)	✗	✓	✗		
mOSAIC	✗	✓	✗		
CSAL (2010)	✗	✗	✓		
OCCI	✓	✓	✗	IaaS PaaS SaaS	
jclouds	✗	✗	✓	IaaS	
Fog	✗	✗	✓		
Deltacloud	✗	✗	✓		
RASIC (2010)	✗	✗	✓		
CDMI	✓	✓	✗		

Therefore, in this research, the STAGER framework is proposed as a step towards overcoming the API synchronization problem. The STAGER provides generic API, which can include management operations or service operations, with semi-automatically generated adapters. In order to evaluate the STAGER framework, we cannot use any of the available generic APIs in the STAGER because they are not open source. Thus, we have proposed service-based generic APIs for blob storage and NoSQl datastore; and the STAGER has been used to generate the adapters for two PaaS platforms; GAE and Windows Azure. The proposed generic-APIs will be used by cloud developers to

implement applications that are independent on heterogeneous PaaS platforms. By this way, the main differences between the currently available research efforts and the STAGER framework are the idea of overcoming the API synchronization problem and it can generate the adapters of any semantically annotated generic API.

References

[1] Dana Petcu and Athanasios V Vasilakos. Portability in clouds: approaches and research opportunities. *Scalable Computing: Practice and Experience*, 15(3):251–270, 2014.

[2] Fotis Gonidis, Anthony JH Simons, Iraklis Paraskakis, and Dimitrios Kourtesis. Cloud application portability: an initial view. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 275–282. ACM, 2013.

[3] Elias Adriano Nogueira da Silva, Renata PM Fortes, and Daniel Lucrédio. A model-driven approach for promoting cloud PaaS portability. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, pages 92–105. IBM Corp., 2013.

[4] Gilbert Pilz Tom Rutt Jacques Durand, Adrian Otto. Cloud Application Management for Platforms Version 1.1, 2014. URL http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf. [Last Accessed: 30 Nov., 2016].

[5] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. TOSCA: portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer, 2014.

[6] Alexander Papaspyrou Thijs Metsch Ralf Nyren, Andy Edmonds. Open Cloud Computing Interface - Core, 2011. URL https://www.ogf.org/documents/GFD.183.pdf. [Last Accessed: 15 Jan., 2017].

[7] Mohamed Sellami, Sami Yangui, Mohamed Mohamed, and Samir Tata. PaaS-independent Provisioning and Management of Applications in the Cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 693–700. IEEE, 2013.

[8] Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. A Case Study for Deploying Applications on Heterogeneous PaaS Platforms. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 246–253. IEEE, 2013.

[9] Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. Implementing Generic PaaS Deployment API: Repackaging and Deploying Applications on Heterogeneous PaaS Platforms. *International Journal of Big Data Intelligence*, 3(4): 257–269, 2016.

[10] E Michael Maximilien, Ajith Ranabahu, Roy Engehausen, and Laura Anderson. IBM altocumulus: a cross-cloud middleware and platform. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 805–806. ACM, 2009.

[11] CMDI. Cloud data management interface (cdmi) version 1.1.1, 2015. URL http://www.snia.org/sites/default/files/CDMI_Spec_v1.1.1.pdf. [Last Accessed: 26 Jan., 2017].

[12] Ilja Livenson and Erwin Laure. Towards transparent integration of heterogeneous cloud storage platforms. In *Proceedings of the fourth international workshop on Data-intensive distributed computing*, pages 27–34. ACM, 2011.

[13] Fawaz Paraiso, Challita Stéphanie, Al-Dhuraibi Yahya, and Philippe Merle. Model-Driven Management of Docker Containers. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, 2016.

[14] Dimitris Zeginis, Francesco D'andria, Stefano Bocconi, Jesus Gorronogoitia Cruz, Oriol Collell Martin, Panagiotis Gouvas, Giannis Ledakis, and Konstantinos A Tarabanis. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. *Scalable Computing: Practice and Experience*, 14(1):17–32, 2013.

[15] Nick Bassiliades, Moisis Symeonidis, Georgios Meditskos, Efstratios Kontopoulos, Panagiotis Gouvas, and Ioannis Vlahavas. A semantic recommendation algorithm for the PaaSport platform-as-a-service marketplace. *Expert Systems with Applications*, 67:203–227, 2017.

[16] Dana Petcu, Beniamino Di Martino, Salvatore Venticinque, Massimiliano Rak, Tamás Máhr, Gorka Esnal Lopez, Fabrice Brito, Roberto Cossu, Miha Stopar, Svatopluk Šperka, et al. Experiences in building a mOSAIC of clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):1, 2013.

[17] Beniamino Di Martino and Giuseppina Cretella. Semantic and algorithmic recognition support to porting software applications to cloud. In *International Workshop on Eternal Systems*, pages 1–15. Springer, 2012.

[18] Giuseppina Cretella and Beniamino Di Martino. Towards automatic analysis of cloud vendors APIs for supporting cloud application portability. In *Complex, intelligent and software intensive systems (CISIS), 2012 sixth international conference on*, pages 61–67. IEEE, 2012.

[19] Giuseppina Cretella and Beniamino Di Martino. A semantic engine for porting applications to the cloud and among clouds. *Software: Practice and Experience*, 45 (12):1619–1637, 2015.

[20] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. Semantic and agnostic representation of cloud patterns for cloud interoperability and portability. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 182–187. IEEE, 2013.

[21] Elisabetta Di Nitto, Marcos Aurélio Almeida da Silva, Danilo Ardagna, Giuliano Casale, Ciprian Dorin Craciun, Nicolas Ferry, Victor Muntes, and Arnor Solberg. Supporting the development and operation of multi-cloud applications: The modaclouds approach. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on*, pages 417–423. IEEE, 2013.

[22] Nicolas Ferry, Marcos Almeida, and Arnor Solberg. The modaclouds model-driven development. In *Model-Driven Development and Operation of Multi-Cloud Applications*, pages 23–33. Springer, 2017.

[23] Fotis Gonidis, Iraklis Paraskakis, and Anthony JH Simons. Leveraging platform basic services in cloud application platforms for the development of cloud applications. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 751–754. IEEE, 2014.

[24] Fotis Gonidis, Iraklis Paraskakis, and Anthony JH Simons. On the role of ontologies in the design of service based cloud applications. In *European Conference on Parallel Processing*, pages 1–12. Springer, 2014.

[25] Ajith Ranabahu, E Michael Maximilien, Amit Sheth, and Krishnaprasad Thirunarayan. Application Portability in Cloud Computing: An Abstraction-Driven Perspective. *IEEE Transactions on Services Computing*, 8(6):945–957, 2015.

[26] E Michael Maximilien, Ajith Ranabahu, Roy Engehausen, and Laura C Anderson. Toward cloud-agnostic middlewares. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 619–626. ACM, 2009.

[27] Nikolaos Loutas, Vassilios Peristeras, Thanassis Bouras, Eleni Kamateri, Dimitrios Zeginis, and Konstantinos Tarabanis. Towards a reference architecture for semantically interoperable clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 143–150. IEEE, 2010.

[28] Zach Hill and Marty Humphrey. CSAL: A cloud storage abstraction layer to enable portable cloud applications. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 504–511. IEEE, 2010.

[29] Luís A Bastião Silva, Carlos Costa, and José Luís Oliveira. A common API for delivering services over multi-vendor cloud resources. *Journal of Systems and Software*, 86(9):2309–2317, 2013.

[30] Stefan Kolb and Guido Wirtz. Towards application portability in platform as a service. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pages 218–229. IEEE, 2014.

[31] David Cunha, Pedro Neves, and Pedro Sousa. PaaS manager: a platform-as-a-service aggregation framework. *Computer Science and Information Systems*, 11(4): 1209–1228, 2014.

[32] Aasim Rafique, Stefan Walraven, Bert Lagaisse, Tom Desair, and Wouter Joosen. Towards portability and interoperability support in middleware for hybrid clouds. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 7–12. IEEE, 2014.

[33] Stefan Walraven, Dimitri Van Landuyt, Ansar Rafique, Bert Lagaisse, and Wouter Joosen. PaaSHopper: Policy-driven middleware for multi-PaaS environments. *Journal of Internet Services and Applications*, 6(1):1, 2015.

[34] Ebtesam Alomari, Ahmed Barnawi, and Sherif Sakr. Cdport: a framework of data portability in cloud platforms. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, pages 126–133. ACM, 2014.

[35] Ebtesam Alomari, Ahmed Barnawi, and Sherif Sakr. Cdport: A Portability Framework for NoSQL Datastores. *Arabian Journal for Science and Engineering*, 40(9): 2531–2553, 2015.

[36] Stefan Kolb and Cedric Röck. Unified cloud application management. In *2016 IEEE World Congress on Services (SERVICES)*, pages 1–8. IEEE, 2016.

[37] Cedric Röck and Stefan Kolb. Nucleus–Unified Deployment and Management for Platform as a Service. *University of Bamberg, Tech. Rep*, 2016.

[38] Antonio Brogi, Jose Carrasco, Javier Cubo, Francesco DAndria, Elisabetta Di Nitto, Michele Guerriero, Diego Pérez, Ernesto Pimentel, and Jacopo Soldani. SeaClouds: An Open Reference Architecture for Multi-cloud Governance. In *Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings*, pages 334–338. Springer, 2016.

[39] Robail Yasrab and Naijie Gu. Multi-cloud paas architecture (mcpa): A solution to cloud lock-in. In *Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on*, pages 473–477. IEEE, 2016.

[40] Chimbili Pallavi and G. S. Udaya Kiran Babu. Developing A Generic Approach to make easy the Developer Task using Multiple Data Stores. *International Journal For Technological Research In Engineering*, 4(2):304–306, 2016.

[41] Arvind Kumar M and T.Auntin Jose. An Generic Cloud Framework for Cloud Based Applications. In *1st International Conference on Innovations in Computing and Networking (ICICN-16)*, pages 502–506, 2016.

List of Publications

Thesis Publications

Article 1:

Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. A Case Study for Deploying Applications on Heterogeneous PaaS Platforms. In *Proceedings of the 2013 International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, FuZhou, China, 2013.

Article 2:

Eman Hossny, Sherif Khattab, Fatma A Omara, and Hesham A Hassan. Implementing generic PaaS deployment API: repackaging and deploying applications on heterogeneous PaaS platforms. *International Journal of Big Data Intelligence*, 3(4):257–269, 2016.

Article 3:

Eman Hossny, Sherif Khattab, Fatma A Omara, and Hesham Hassan. Semantic-based generation of generic-API adapters for portable cloud applications. In *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms*, pages 1–5. ACM, 2016.

Article 4:

Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. Towards a Standard PaaS Implementation API: A Generic Cloud Persistent-Storage API. *International Journal of Cloud Computing*. [Submitted for publication].

Article 5:

Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. STAGER: A framework for semantic-based generic-API adapters generation for portable cloud applications. [Submitted for publication].

Other Publications

Article 1:

Eman Hossny, Sara Salem, and Sherif Khattab. Towards automated user-centric cloud provisioning: Job provisioning and scheduling on heterogeneous virtual machines. In *Informatics and Systems (INFOS), 2012 8th International Conference on*, pages CC–18. IEEE, 2012.

Article 2:

Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. Finding the pin in the haystack: A Bot Traceback service for public clouds. In *Intelligent Computing and Information Systems (ICICIS), 2015 IEEE Seventh International Conference on*, pages 258–262. IEEE, 2015.

[pages=-]./Chapters/arabic$_c$*over.pdf*