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 ITAC from 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

Declaration of Authorship i

Acknowledgements ii

Abstract iii

List of Figures v

List of Tables vi

Abbreviations vii

1 Literature Review 1
  1.1 Standardization Efforts ................................. 2
  1.2 Research Projects ...................................... 5
  1.3 generic APIs ............................................. 8
  1.4 Discussion ............................................... 13

References 16
List of Figures
List of Tables

1.1 Research Efforts for Vendor Lock-in Problem . . . . . . . . . . 14
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCORDS</td>
<td>Advanced Capabilities for Compatible One Resource Description System</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large OBjects</td>
</tr>
<tr>
<td>CORDS</td>
<td>Compatible One Resource Description System</td>
</tr>
<tr>
<td>CAPEX</td>
<td>CAPital EXpenses</td>
</tr>
<tr>
<td>CBSS</td>
<td>Cloud Blob Storage Service</td>
</tr>
<tr>
<td>CF</td>
<td>CloudFoundry</td>
</tr>
<tr>
<td>COAPS</td>
<td>Compatible One Application and Platform Service</td>
</tr>
<tr>
<td>CDMI</td>
<td>Cloud Data Management Interface</td>
</tr>
<tr>
<td>DAML</td>
<td>DARPA Agent Markup Language</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>GAE</td>
<td>Google App Engine</td>
</tr>
<tr>
<td>GCD</td>
<td>Google Cloud Datastore</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Not only Structured Query Language</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OCCI</td>
<td>Open Cloud Computing Interface</td>
</tr>
<tr>
<td>OPEX</td>
<td>OPerating EXpenses</td>
</tr>
<tr>
<td>OS</td>
<td>OpenShift</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PADM</td>
<td>PaaS Application Description Model</td>
</tr>
<tr>
<td>PBS</td>
<td>Platform Basic Services</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource Description Framework Schema</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SAVG</td>
<td>Semi-automatic Adapter Validation Generation</td>
</tr>
<tr>
<td>SCOP</td>
<td>Source Code Ontology Population</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Simple Protocol And RDF Query Language</td>
</tr>
<tr>
<td>STAGER</td>
<td>SemanTic-based GenERation of Generic-API Adapters</td>
</tr>
<tr>
<td>Std-PaaS APIs</td>
<td>Standard Platform as a Service Application Programming Interface</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
</tr>
</tbody>
</table>
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 require 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 Amazon 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 have 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] provide a 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 problems. 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 abstrac-
tion 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 applica-
tions 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 frame-
work 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 pro-
vide an adapter for each one of the supported databases. They have tested their frame-
work 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 Sim-
plesDBAdapter). 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 pro-
vides a platform and unified API to manage and deploy service-based cloud applica-
tions 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 provides Java-based generic APIs as an open source for three services; compute, storage, and load balancer. Fog is similar to jclouds but it is based on Ruby. Libcloud provides python-based unified API for compute and DNS services. Deltacloud 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

<table>
<thead>
<tr>
<th>Research Effort</th>
<th>Standard</th>
<th>Generic Manag. API</th>
<th>Generic Implem. API</th>
<th>Cloud Model</th>
<th>API Synchronization Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMP (2014)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOSCA (2014)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COAPS (2013, 2016)</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud4SOA (2013)</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nucleus (2016)</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaaS Manager (2014)</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDPort (2014, 2015)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>PaaS</td>
<td>All Adapters are suffering from API synchronization problem</td>
</tr>
<tr>
<td>PaaSShopper (2014, 2015)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDCP (2013)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODBAPI (2016)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDMI-proxy (2011)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SeaClouds (2016)</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>IaaS</td>
<td></td>
</tr>
<tr>
<td>MCPA (2016)</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>PaaS</td>
</tr>
<tr>
<td>mOSAIC</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSAL (2010)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCCI</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>IaaS</td>
<td>PaaS</td>
</tr>
<tr>
<td>jclouds</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fog</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>IaaS</td>
</tr>
<tr>
<td>Deltacloud</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RASIC (2010)</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDMI</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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


References


List of Publications

Thesis Publications

Article 1:


Article 2:


Article 3:

Article 4:


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:


Article 2:


References

[pages=-]./Chapters/arabic_over.pdf