Cloud Application Portability: An Initial View

Fotis Gonidis
South East European Research Centre
International Faculty of the University of Sheffield
24 Proxenou Koromila street
Thessaloniki, 54622, Greece
fgonidis@seerc.org

Iraklis Paraskakis
South East European Research Centre
International Faculty of the University of Sheffield
24 Proxenou Koromila street
Thessaloniki, 54622, Greece
iparaskakis@seerc.org

Anthony J. H. Simons
Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
Sheffield S1 4DP, United Kingdom
A.Simons@dcs.shef.ac.uk

Dimitrios Kourtesis
South East European Research Centre
International Faculty of the University of Sheffield
24 Proxenou Koromila street
Thessaloniki, 54622, Greece
dkourtesis@seerc.org

ABSTRACT

Growing interest towards cloud application platforms has resulted in a large number of platform offerings to be already available on the market and new related products to be continuously launched. However, there are a number of challenges that prevent cloud application platforms from becoming widely adopted. One such challenge is application portability. This paper reports on an ongoing effort to explore the area of cloud application portability. We briefly examine the issue of heterogeneity in cloud platforms and highlight specific platform characteristics that may hinder the portability of cloud applications. We present some high level approaches and existing work that attempts to address this challenge. In order to narrow down the area of our exploration we have been carrying out an experiment in cross-platform application development and deployment with four prominent cloud platforms: OpenShift, Google App Engine, Heroku, and Amazon Elastic Beanstalk. We briefly discuss our initial conclusions from this ongoing experimentation.

Categories and Subject Descriptors

D.2.7 [Distribution, Maintenance, and Enhancement]: Portability

General Terms

Documentation, Experimentation

Keywords

Cloud platforms, PaaS, application portability, Standardization, Intermediation

1. INTRODUCTION

Cloud computing is a relatively new paradigm that promises to revolutionize the way IT services are provided and consumed. There are multiple benefits that companies can gain from adopting the cloud computing model [1]. These benefits differ with respect to the particular type of cloud service involved, namely, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

Particularly in the case of PaaS, a key benefit is that users can develop and deploy applications without the burden of setting up and maintaining the necessary programming environment and infrastructure that the application is executed on. In addition, most platforms offer tools and services that help developers to decrease development time. Moreover, some PaaS offerings allow Independent Software Vendors (ISVs) to make their applications available on the platform's marketplace reaching a large number of potential customers.

However, different cloud application platform offerings are characterized by considerable heterogeneity. Because of incompatibilities, users that develop applications on a specific platform may encounter significant problems when trying to deploy their application in a different environment. This gives rise to the familiar problem of vendor lock-in [2], which has been a challenge long before the advent of cloud computing.

Consumers need to be able to easily change between cloud providers and should be free to choose the one that better serves their needs in terms of quality and/or cost. The ability of consumers to switch from one cloud platform provider to another can be critical for their business, especially when a cloud provider's operation is unexpectedly terminated. A real example to illustrate this argument is the case of Coghead [3], an online application development platform supporting the development and hosting of data-driven applications. The platform had managed to attract hundreds of developers before it suddenly announced that it would stop operating, calling all customers to export the data that was stored in their applications, but not giving them the option to port the actual applications to some other platform.

For developers to be able to exploit the full advantages of PaaS, they should be able to deploy their cloud applications across multiple platforms, without lock-in to a particular vendor. To achieve this, a new approach to cloud application development must be adopted. The key concept is for users not to develop

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
BCI'13, September 19-21, 2013, Thessaloniki, Greece.
Copyright 2013 ACM 978-1-4503-1851-8/13/09 ...$15.00.

applications directly against proprietary platform environments. Rather, developers should use either standard and widely adopted technologies, or abstraction layers which decouple application development from specific target platforms. Ensuring portability across cloud providers would eliminate the vendor lock-in problem and would allow consumers to switch between vendors according to their needs. In turn, this would increase consumers' trust towards cloud computing and public cloud services.

This paper focuses on the challenge of application portability in the scope of Platform as a Service environments. In this context we examine how the heterogeneity of cloud platforms may hinder cross-platform deployment of an application. We believe that there is no universal solution to this problem; addressing the issue of application portability across platforms that don't share any common characteristics is practically impossible. Consequently, we attempt a high-level classification of cloud application platforms and thereafter we narrow down on a specific category of platforms where our interest will focus on. Next, we discuss specific conflicts that may occur when deploying an application to different cloud platforms of the same category. Thereby we attempt to illustrate potential points where our research work may subsequently focus on. In section 3 we mention existing approaches and concrete research work attempting to tackle application portability. Finally, in section 4 we discuss preliminary results from an on-going experiment in cross-platform development and deployment of a cloud application. The experiment involves four widely used cloud application platforms and is aimed at helping us gain a deeper understanding of the characteristics of the problem in a realistic setting.

2. PORTABILITY IN THE CONTEXT OF PLATFORM AS A SERVICE

2.1 Overview of cloud platforms

Growing interest towards cloud application platforms has resulted in a large number of platform offerings to be already available on the market and new platform products to be continuously launched. The platforms available on the market form a wide spectrum of solutions that a user can choose from. These solutions may significantly vary from each other. To allow for better understanding of the types of cloud platform offerings available today, and to highlight the differences between them, we present a brief overview of a few illustrative examples. This section is part of an on-going survey on cloud platforms that is due to be published in the coming months.

2.1.1 Cloud platforms

The purpose of this section is to present some instances of different types of cloud platforms available today. The examples provided in the following paragraphs represent popular Platform as a Service offerings supporting different approaches to the development and deployment of applications. The selection was based on the fact that the authors have working experience with the following platforms and also the presented offerings are prominent solutions of the cloud platform ecosystem.

OpenShift: OpenShift is a cloud platform managed by Red Hat. It provides several programming languages and frameworks that a developer can choose from to create an application, like Java, PHP, Ruby, Python etc. Regarding the database offerings, OpenShift provides the widely used relational MySQL and PostgreSQL as well as MongoDB, a noSQL document-oriented database. The platform does not offer any storage service. The user can create the application using a traditional development tool locally. OpenShift provides a command line tool to be used for deploying the application on the platform. Alternatively, the application can be developed in Eclipse IDE and deployed on the platform with the help of a deployment plug-in for Eclipse provided by OpenShift. OpenShift is a generic cloud platform meaning that there is no restriction on the scope or type of the applications that one can build and deploy. A user can execute any source code as long as it is compatible with the platform. The platform does not require platform-specific libraries to be used and does not offer any standard application logic pre-packaged in the form of native services (e.g. application authentication logic). Therefore development time is relatively high, since developers need to code all the application functionality from the ground up. OpenShift does not provide native integration with 3rd party applications.

Google App Engine: Google App Engine (GAE) is a cloud platform offered by Google. Developers can code their application in Java, Python or Go - an open source programming language developed by Google[1]. Regarding database support, GAE provides a traditional SQL database called Google Cloud SQL, and a noSQL database called App Engine Datastore. Apart from database services, GAE also provides a file storage service. Developers deploy applications locally and deploy them to the platform via a command line tool or the Google plug-in for Eclipse. To speed up development and enhance application functionality Google provides API integration with a wide range of its own products like Google Docs, Google Maps for location-based applications, Google Wallet for online payments, etc. However Google App Engine has still a generic application scope. That means that users are free to deploy their own source code provided that it complies with the restrictions of the platform.

Zoho Creator: Zoho Creator is the cloud platform offered by Zoho. The platform is focused on the development of office and CRM applications that are data oriented. It follows a different application development paradigm than the previous mentioned platforms. In Zoho Creator, developers are not expected to create their applications using a programming language. Instead, applications are developed via the web-browser using a visual design interface. The platform offers a toolkit that includes design forms to be used to create data fields, data analytics and reports as well as a scripting language called Deluge, for defining workflows, business rules, etc. The goal of the tools that are offered by the platform is to drastically reduce the time and level of expertise required to develop applications. In contrast to other cloud application platforms such as OpenShift and Google App Engine, the scope of the applications to be developed on Zoho Creator is rather narrow, limited to the tools and templates that are already available. Unlike the previously mentioned platforms, users are not able to upload their own source code.

2.1.2 Cloud platforms classification

OpenShift, Google App Engine and Zoho Creator can be considered representative examples of three different categories of cloud application platforms.

The first category includes the platforms that adopt or provide support for standard, widely adopted and used technologies, like programming languages and databases. Platforms in this category have a generic application scope and users can upload the source

[1] http://golang.org/project/

code of their application. They don't provide native APIs for offering custom functionality, increasing this way the application development time. However the fact that they offer only standard programming technology without native APIs maximizes the portability of the application.

The second category includes platforms that also offer standard programming languages and databases like Java and MySQL respectively. However, in order to decrease the application development effort they also offer custom functionality via native APIs. The degree of the lock-in effect is determined by whether or not a developer will choose to speed up development by making use of the custom functionality.

The third category includes platforms that adopt a different application development paradigm, characterized by tools for online development via a web browser, using visual interfaces and design templates. Developers are provided with a generic visual application development framework that they can customize to meet their requirements. These platforms have a specific application scope that is oriented in CRM systems and similar data-driven business applications. Development time can be dramatically decreased due to the automated development processes. However this is done at the expense of portability and the limited application scope.

It becomes obvious that there can be significant variations between cloud platform offerings that are available on the market. Due to the heterogeneity between offerings it is not feasible to tackle application portability by engineering a solution applicable to the whole wide spectrum of available cloud platforms. Efforts need to be concentrated on a specific set/class of platforms that present similar characteristics.

The first category of cloud application platforms consists of offerings that are strongly characterised by the use of standard and widely adopted technologies. Therefore the lock-in effect in these cases is not significant. On the other side of the spectrum, the third category of platforms comprises offerings where developers are primarily concerned with minimizing the effort of application development. In this case the lock-in factor is very high and cross-platform deployment is practically impossible, but this seems to be a trade-off that developers are willing to accept.

For these reasons, the initial focus of our research work is on the second category of cloud platforms, namely the ones that offer proprietary services via native APIs (e.g. for file storage or data storage) and allow developers to develop and deploy arbitrary source code.

After narrowing down the research focus on a specific category of platforms, the next step is to define the exact part of the cloud application where our research efforts will be focused on. From the description of Google App Engine, certain platform characteristics emerge that could potentially hinder application portability: programming languages and frameworks, data stores, and platform-specific services. These characteristics are addressed in some detail in the next section.

2.2 Portability issues in cloud applications

To proceed with our investigation into the problem of cloud application portability we need to identify specific points of conflict emerging when attempting to deploy an application to multiple platforms. In other words, we need to identify which aspects of a cloud application may be addressed differently by cloud platforms. In this section we discuss the following four potential conflict points: programming languages and frameworks, platform-specific services, data stores and platform specific configuration files.

a) Programming languages and/or frameworks

The specific programming languages and frameworks that an application has been built with is obviously a major determinant for cross-platform deployment. Each cloud platform supports certain languages, frameworks, and versions thereof. For example, while Google App Engine (GAE) provides support for Java it does not support the same extent of standard Java class libraries supported by OpenShift.

b) Platform specific services

An important characteristic of several cloud platforms is that they provide certain services via specific APIs. A service can be considered as high-level functionality that the provider can use without the need to implement it from scratch. Such examples are analytic tools for handling data sets, APIs for image manipulation etc. Developers can drastically reduce the application development time by using such platform services. Instead of programming every bit of functionality from the ground up, they can integrate it into their application by binding to the respective platform APIs.

For example, consider the application in Figure 1. It comprises functionality for reading-in data from a database and producing some analytics. It performs logging and monitoring of the resources that it consumes. Moreover it alerts the stakeholders about various events via e-mail and SMS. These blocks of functionalities can be provided by the platform as services through APIs. The developer only needs to "glue" the offered services together in order to build the application. This is only an example application for allowing the reader to gain a better understanding of the platform specific services. Each platform provider may offer a wider or smaller range of such specific services.

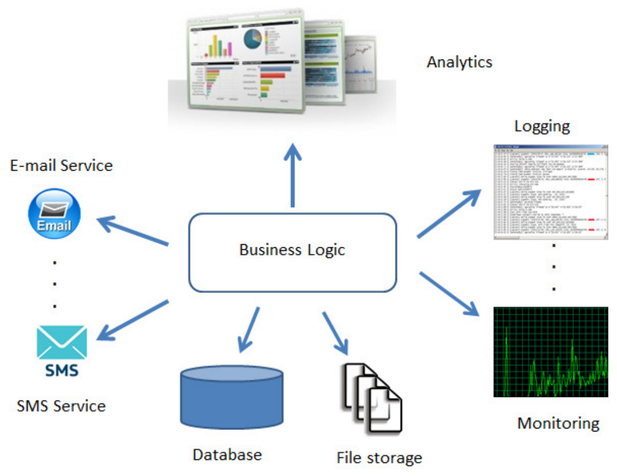

Figure 1. Example of application synthesized by multiple platform services

Let us assume that a developer chooses a certain platform in order to develop and deploy the above mentioned application. A portability issue arises when the application needs to be ported to a different cloud platform. There are two cases:

1) The target platform doesn't provide the full set of services that the application uses. For example, SMS

services and monitoring services are not supported. In this case the developer would need to recreate the missing functionality from scratch on the new target platform.

2) The target platform supports the services that the application uses but provides different APIs in order to use them. In this case the developer would need to modify the application code and align it with the APIs of the new target platform.

In both cases, the application cannot directly be ported across multiple platforms. The developer needs to modify the application in order to be deployable to different platforms.

c) Data Storage

Data storage is an essential part of an application. There are two types of data storage: database stores and file stores. The first one is used for storing structured data while the second one could be perceived as an analogy to a hard disc drive on the cloud.

Almost every modern cloud application needs to access data from a database. A high level classification can be made into SQL and noSQL databases:

- **SQL database:** This type of database represents the widely used, traditional relational database. All major cloud platforms offer relational database as a service. Specific examples are: Amazon Relational Database Service (Amazon RDS), Google Cloud SQL, and Windows Azure SQL Database.

- **noSQL database:** noSQL database is a relatively new category of databases compared to SQL. The noSQL term groups together all database systems that don't adhere to the relational structure. Main characteristics of these systems, according to R. Catell [4] and Cure et al. [5] are: the ability to distribute data over many servers, the simple operations compared to the complex SQL queries and joins, the ability to dynamically add new attributes to data entities, and the fast access times for storage, data retrieval and analysis. There is a wide variety of noSQL database types as listed in the work of R.Catell [4] and Burtica et al. [6]. As examples we can consider Key-value Store (Redis, Dynamo etc), Document store (MongoDB, SimpleDB, etc.), Graph Store (Neo4j etc.)

Different platforms often support different types of database. Therefore the following conflicts may arise when trying to port an application across various platforms:

1) **Incompatible data structures**: As it became clear there is a wide range of available databases where each one of them adopts a different data structure. Portability issues are bound to arise when trying to move data from a SQL to a noSQL database, but also between different types of noSQL databases, e.g. when moving from a key-value store to a document store.

2) **Different query languages**: Apart from the incompatibility due to different data structures, conflicts may occur at the way of querying data. Databases use their own APIs or query languages. Therefore even when data is moved across databases of the same category (e.g. a document store), portability issues may arise concerning the way the database is accessed.

3) **Data migration (export/import formats)**: Another issue that should be considered is data migration. It may happen that an exported database cannot directly be imported to another database engine due to incompatible data formats.

In addition to issues around data storage, points of conflict may also arise in relation to the file storage services offered by different cloud platforms. A file store service can be provisioned via two ways:

- Use of graphical interface. Human users of the cloud application can manually perform operations on the file storage space.
- Use of APIs. Platform APIs can be used by the cloud application to obtain programmatic access to the file storage.

In the latter case each platform vendor provides a custom proprietary API in order to allow applications interact with the storage space. Therefore when an application is ported across different platforms the storage API needs to be adjusted accordingly to fit the host provider. As a result a portability issue is raised. Major file storage services are: Amazon Simple Storage Service by Amazon and Google Cloud Storage by Google.

d) Platform specific configuration files

Similar to the configuration files that traditional software applications require in order to instruct the hosting environment on how to execute the applications, cloud platforms may require analogous configuration files. For example Google App Engine uses the "appengine-web.xml" file. The process of adapting the configuration files to each target cloud platform adds to the overall overhead of cross-platform deployment of a cloud application.

In this section the multiple conflict points between cloud platforms were highlighted. There are several research works attempting to address the challenge of portability. Some representative ones are presented in the next section.

3. EXISTING WORK

There are two generic approaches that could be adopted in order to overcome the incompatibilities between cloud platforms, and eventually ease application portability: standardization and intermediation.

Standardization implies the definition of common set of standards for PaaS offerings. The adoption of such standards by all cloud providers would enable developers to create and manage their applications independently of specific platform environments and then deploy them to the cloud platform of their choice. This set of standards could include a standardized API to access the service offered by the platform and to manage the deployment and the lifecycle of the application. The National Institute of Standards and Technology (NIST) has published a roadmap [7] about cloud computing standards pinpointing what interfaces need to be standardized in each cloud computing service level (IaaS, PaaS, SaaS).

There are several active standardization organizations. The Distributed Management Task Force (DMTF) has launched the Open Virtualization Format (OVF) [8] in an attempt to standardize the VMs format and enable their portability. The Storage Networking Industry Association (SNIA) has created the Cloud Data Management Interface (CDMI) [9] as an attempt to standardize access to cloud storage services. Open Cloud Computing Interface (OCCI) [10] is active in standardizing the way VMs are managed. Topology and Orchestration Specification for Cloud Applications (TOSCA) [11] is a standard supported by OASIS aiming at standardizing the packaging of the application in order to enable automatic cross-platform deployment.

Standardization seems to be a very efficient approach to achieve cloud portability. However, for reasons not necessarily related to technology, it is very difficult for all cloud platforms to eventually agree on a common set of standards. All major cloud vendors use proprietary APIs and file formats as a way of locking-in customers to their services. The effort required to re-engineer an application in order for it to be ported to another platform is discouraging customers from moving. In addition, a set of common standards would prevent platform providers from offering the special, platform-specific features that allow vendors to differentiate from their competitors.

Another approach towards achieving portability between platforms is intermediation. That is, introducing an intermediate layer that decouples application development from specific platform APIs and supported formats. In this case developers create their applications using an intermediate API which is platform agnostic and which can "hide" or "wrap" the proprietary APIs of particular vendors. The intermediate layer prevents developers from being bound to specific programming languages or data stores. For example an application could be developed in a language-independent manner, and later on, through model transformations, be translated into the particular programming language supported by a PaaS provider (such as Java, Python or C#), the database query language particular to a platform database (e.g. MySQL or noSQL databases) or file storage API (e.g Amazon S3 or Azure blobstore).

jClouds is an open source library that can be used by application developers in order to abstract cloud vendors' specific APIs. jClouds offers a file storage service. It allows an application to store and read files from a remote store provided by a cloud provider. The storage service of jClouds (blobstore) consists of the following structure: Container, which is the top level directory, Blobs, which contain the data to be stored and Folders which are used to organize the blobs

Major file storage services that jClouds can abstract are Azureblob by Microsoft Azure and Amazon S3 by Amazon. jClouds API is offered in two programming languages: Java and Closure.

Another example of intermediation approach is mOSAIC [12]. mOSAIC is an open source platform which promotes application portability. It achieves this by implementing multiple API layers which gradually offers the developers an abstraction from the native APIs. mOSAIC API supports the use of cloud databases, cloud file storages and communication service.

Regarding the abstraction of the database store, Cure et al. [5] put forward an approach for providing abstract access to non-relational database systems. The focus of this work lies in allowing developers accessing the noSQL databases using the familiar syntax of SQL.

The architecture of the proposed approach consists of two parts:

- Translation of the SQL query into an intermediate query language, BQL (Bridge query language). SQL is a declarative query language , while, as it is in mentioned in [5], most noSQL follow a procedural query approach. Therefore BQL is introduced as an intermediate step in order to bridge SQL with noSQL languages.
- Translation of BQL into specific noSQL query. The second step is to transform the BQL query to the native query that is supported by the source database system.

Apart from the use of APIs, the issue of cloud portability can be addressed by exploiting Model Driven Engineering (MDE) techniques [13]. MDE is an approach to system and software development in which software models play an indispensable role [14]. MDE is based on two core ideas: Abstraction and Automation. Abstraction enables decoupling application development from targeting specific platforms. Automation refers, among others, to the ability to change the level of abstraction automatically, using model transformations. Model transformations can automate the process of generating platform specific implementations. MDE has been since many years in practice for developing traditional software systems. The most prominent and widely used modeling language for that purpose is UML[2]. In recent years, with the emergence of cloud computing and cloud application development, efforts are made in exploiting the benefits of MDE in creating portable applications. Similar to the traditional software development, the goal is to abstract the cloud application design and development from targeting specific platforms. The creation process begins with building a platform independent model (PIM) and then using automated model transformations translate the PIM into a platform specific model (PSM) targeting particular cloud platforms (Figure 2).

Ajith Ranabahu et al. put forward an abstraction driven approach to achieve application portability [15]. Particularly they have been working on a Domain Specific Language (DSL) called MobiCloud [16].

MobiCloud is a modeling language that closely resembles the Model-View-Controller (MVC) design by providing constructs for each of the three key components: model, view, and controller. This approach allows developers to create simple CRUD mobile applications using a graphical editor and a scripting language. The platform automatically generates the source code for uploading the backend of the application on Google App Engine and Amazon EC2.

To the best of our knowledge, there has been no extensive survey so far that has undertook to describe in detail the problem space of cloud application portability and how solutions like the ones mentioned above are mapped to that space. Such a study is essential in order to understand the root causes for the platform lock-in effect and the desirable characteristics of solutions to this problem. To that end, we have designed an experiment with a test case application and four target cloud application platforms, presented in the next section.

[2] http://www.uml.org/

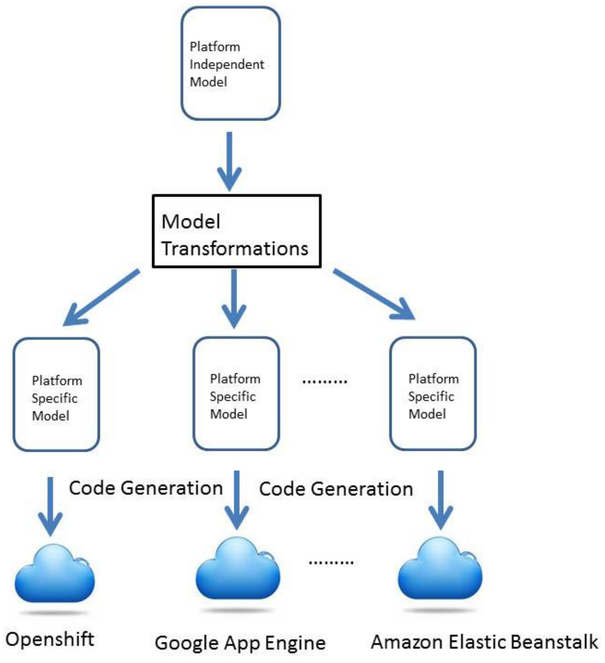

Figure 2. MDE approach in developing cloud portable applications

4. EXPERIMENTATION WITH CLOUD PLATFORMS

Our research focus lies in addressing the issue of application portability across cloud platforms. As it was described in 2.2 there are multiple conflict points where a portability issue could be raised within a cloud application like data stores or platform specific services. In order to get an insight and be able to narrow down our research focus to a specific context, it is essential that we experiment with a realistic application.

Towards identifying the commonalities and incompatibilities among the cloud platforms, a toy application has been developed. A number of target platforms have been selected from category one and two for the application to be deployed on. Particularly, the following four platforms have been selected: OpenShift, Google App Engine, Heroku, and Amazon Elastic Beanstalk. All four platforms were chosen because they are popular solutions in the domain of cloud computing and the authors are familiar with those. The list of the target platforms is not exhaustive. There are several other cloud platforms such as Windows Azure. However the purpose of this experimentation is not to test exhaustively all available cloud platforms but rather to draw an initial high level conclusion about portability of simple applications across certain prominent platforms.

We started with the development of a very simple application without using any platform specific technology and APIs. The initial goal was to examine whether there are cross-platform deployment issues when the application consists only of standard, widely used technologies such as JavaEE and MySQL. Therefore in relation with the four conflict points that were discussed in section 2.2 (programming language and frameworks, platform specific services, data stores and configuration files) the platforms will be evaluated against the support of the programming framework, the database and the required configuration files if any.

The application is going to be enhanced gradually with platform specific services and we will re-examine the feasibility of the application portability. This will be an iterative process until we reach a mature level of understanding of the examined platforms that will lead us to the exact definition of our research scope.

4.1 Description of the test application

The test application that was developed and deployed in the above mentioned cloud platforms is a simple application that allows users to perform “create”, “read” “update” and “delete” operations on certain entities. Representative view of the user interface is shown in Figure 3.

Initially the application was developed and deployed on a local workstation. The development framework was Java EE and the deployment facility was a JBoss AS 7.1 application server.[3] For building the presentation layer, the JavaServer Faces[4] (JSF) framework together with Primefaces[5] library was used. The business logic layer has been built using Enterprise Java Beans[6] (EJB) technology. For accessing the data layer, JPA[7] was used, together with Hibernate. Data is stored in a MySQL 5.1 database.

4.2 Discussion

In this section we discuss the initial results that were obtained from the deployment of the test application in the four platforms. We specifically examine the support of the four platforms for the JavaEE framework and MySQL database. The process of deploying the application in the four target platforms was divided in 4 phases: Deployment of the presentation layer, business logic layer, data access layer and data migration. For each of the 4 phases we comment on the modifications that were required, if any.

As it was mentioned in section 4.1, JSF and Primefaces were used for the presentation layer. OpenShift supports the JSF specification. Primefaces framework could easily be loaded by including the respective library when deploying the application. on the platform.

When the application was ported on Google App Engine, some modifications were required. The platform supports specific version of the JSF and therefore we were required to upload the respective library versions. Furthermore the configuration file needed to be adjusted accordingly. No issues were encountered concerning Heroku and Amazon Elastic Beanstalk. JSF and Primefaces were loaded, simply by adding the library files without further configuration.

Regarding the business logic layer, Java EJBs were used. OpenShift supports the Java EE specification. Therefore there were no complications when deploying the business logic layer on the platform. That was an expected result since OpenShift is using

[3] http://www.jboss.org/

[4] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

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

[6] http://www.oracle.com/technetwork/java/javaee/ejb/index.html

[7]http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

JBoss application server, the same as we used for the development of the application. GAE, although it provides some support for Java EE features, doesn't support the full Java EE specification. EJBs are among the features that are not supported. Therefore we needed to modify the business logic layer and remove the EJBs in order to deploy it on the platform. To the best of our knowledge, Heroku and AEB don't also natively support EJBs specifications. Consequently the application needed to be modified.

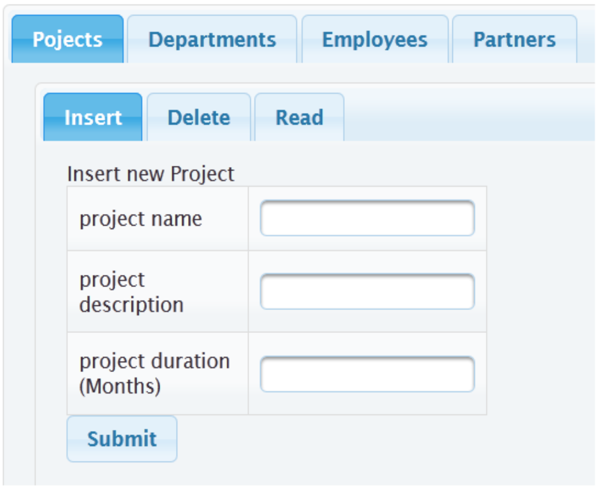

Figure 3. User Interface view of the test application

Concerning the data access layer, no particular issue was encountered. All four platforms support JPA specification. In OpenShift, Heroku and AEB, Hibernate was used as implementation framework, while in the case of GAE EclipseLink was deployed.

Regarding the database store, each platform provides an implementation of MySQL database. In OpenShift, users can import and export a dump file of the database using the open source tool PhPMyAdmin which is offered by the platform. GAE provides the Google SQL database, which supports most of the SQL statements. Users can create an instance of Google SQL database and then import data using a graphical interface. Heroku natively offers only PostgreSQL database. However there are third-party MySQL implementations that can be used. We chose to connect to Amazon RDS, which is Amazon's MySQL offering. It is first required that user creates an Amazon RDS instance and then connect their application with the database. Importing data in Amazon RDS instance can be done via a database administration tool. In AEB, as it is just stated, Amazon RDS was used.

As it was stated in the beginning of section 4, the test application was developed using standard and widely used technologies. Given the simplicity of the application functionality and the fact that no custom platform functionality was used, the anticipated result was that the application would be directly portable across the four target platforms. However, even at this initial stage, certain portability issues were encountered. Particularly, the conflicts were raised at the level of the programming frameworks and the configuration files. Not all platforms supported EJB specifications. Therefore we needed to modify the application removing EJBs. Furthermore, in the case of Google App Engine, certain configuration files needed to be adjusted in order to deploy the database and the JSF specification.

Although the majority of the target platforms were chosen from the second category of our classification, we would not anticipate any different results were the platforms chosen from the first category. The reason is that at this initial stage of our experimentation no platform specific functionality was used. Furthermore as it was mentioned in the section 2.1.2, similar to the platforms in the first category, the offerings in the second one also support standard and widely used technologies. The result would not be the same if platform offerings from the third category were chosen. The reason is that in the third category even simple applications are developed using proprietary tools and technologies.

As future steps, the application is going to be enhanced with more complex functionality and cross-platform deployment will be re-evaluated. Particularly, we are planning to use platform specific services which are offered by a set of cloud platforms of the second category. Examples of such cloud services are message queue and billing service. We expect to encounter further portability issues when dealing with platform specific services. The issues may be raised due to incompatible APIs, differences in the architecture of the offered services, differences in the supported operations, etc.

5. CONCLUSIONS

In this paper we discussed the challenge of cloud application portability in the context of Platform as a Service offerings. Due to the high degree of heterogeneity between cloud application platforms, any approach to tackle application portability needs to target a specific set/class of platforms that present similar characteristics. In this context we attempted a high level classification of cloud platforms into three categories.

The first category includes platforms that support standard and widely used technologies without offering platform specific functionality via proprietary APIs. Platforms included in the second category also support standard technologies for application development, but also provide custom functionality via native APIs. The third category includes platforms that don't support standard programming technologies. As we move from one category to the next, the portability of applications is gradually decreased.

Next, we focused on specific characteristics of platforms in the second category that may hinder application portability, such as programming languages and frameworks, database offerings, file storage service, platform specific services and configuration files.

We presented related work aimed at addressing the challenge of cloud application portability and distinguished between two generic approaches to tackle the issue of application portability: standardization and intermediation. Standardization addresses cross-platform portability through the adoption of common standards by cloud providers. Alternatively intermediation enables developers to create applications independently of a specific platform and then bind them to particular target platforms through some form of automatic translation.

Finally, we reported on an ongoing experiment that we have been carrying out on cross-platform application development and deployment with four prominent cloud platforms: OpenShift, Google App Engine, Heroku and Amazon Elastic Beanstalk.

6. ACKNOWLEDGEMENTS

This research work is funded by the FP7 Marie Curie Initial Training Network "RELATE".

7. REFERENCES

[1] F. Gonidis, I. Paraskakis, and D. Kourtesis, "Addressing the Challenge of Application Portability in Cloud Platforms," in *7th South-East European Doctoral Student Conference*, Thessaloniki, 2012, pp. 565–576.

[2] Neal Leavitt, "Is Cloud Computing Really Ready for Prime Time?" *Computer*, vol. 42, no. 1, pp. 15–20, Jan. 2009.

[3] Jason KinCaid, "Coghead Grinds To A Halt, Heads To The Deadpool," *techcrunch*, Feb-2009. [Online]. Available: http://techcrunch.com/2009/02/18/coghead-grinds-to-a-halt-heads-to-the-deadpool/:Last Accessed June 26th 2013.

[4] R. Cattell, "Scalable SQL and NoSQL data stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.

[5] O. Curé, R. Hecht, C. Le Duc, and M. Lamolle, "Data Integration over NoSQL Stores Using Access Path Based Mappings," in *Database and Expert Systems Applications*, vol. 6860, A. Hameurlain, S. Liddle, K.-D. Schewe, and X. Zhou, Eds. Springer Berlin / Heidelberg, 2011, pp. 481–495.

[6] R. Burtica, E. M. Mocanu, M. I. Andreica, and N. Tapus, "Practical application and evaluation of no-SQL databases in Cloud Computing," in *Systems Conference (SysCon), 2012 IEEE International*, 2012, pp. 1 –6.

[7] Michael D. Hogan, Fang Liu, Annie W. Sokol, and Tong Jin, "NIST Cloud Computing Standards Roadmap," NIST, SP500-291-v1.0, Aug. 2011.

[8] "Open Virtualization Format Specification," Distributed Management Task Force, DSP0243, Jan. 2013.

[9] "Cloud Data Management Interface (CDMI^{TM})," Storage Networking Industry Association (SNIA), Version 1.0.2, Jun. 2012.

[10] Thijs Metsch and Andy Edmonds, "Open Cloud Computing Interface - Infrastructure," Open Grid Forum, GFD-P-R.184, Jun. 2011.

[11] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable Cloud Services Using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80 –85, May 2012.

[12] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable Cloud applications—From theory to practice," *Future Generation Computer Systems*, Jan. 2012.

[13] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds," in *2012 ICSE Workshop on Modeling in Software Engineering (MISE)*, 2012, pp. 50–56.

[14] Bran Selic, "MDE Basics with a UML Focus," presented at the 12th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Model-Driven Engineering, Bertinoro, Italy, Jun. 2012.

[15] A. Ranabahu, E. M. Maximilien, A. Sheth, and K. Thirunarayan, "Application Portability in Cloud Computing: An Abstraction Driven Perspective," *IEEE Transactions on Services Computing*, vol. 99, no. 1, p. 1, Apr. 2013.

[16] A. H. Ranabahu, E. M. Maximilien, A. P. Sheth, and K. Thirunarayan, "A domain specific language for enterprise grade cloud-mobile hybrid applications," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, &#38; VMIL'11*, New York, NY, USA, 2011, pp. 77–84.