Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 50 (2015) 73 – 80

Procedia Computer Science

2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Framework for Platform Agnostic Enterprise Application Development Supporting Multiple Clouds

Aparna Vijaya[a], Neelanarayanan V[b]

[a]*Assistant Professor, Vellore Institute of Technology Chennai Campus, Chennai 600048, India*
[b]*Associate Professor, Vellore Institute of Technology Chennai Campus, Chennai 600048, India*

Abstract

Rapid development of Cloud Computing and its increasing popularity in recent years has driven many commercial cloud providers in the market. Cloud service providers have a lot of heterogeneity in the resources they use. They have their own servers, different cloud infrastructures and APIs and methods to access the cloud resources. Lack of standards has caused the collaboration and portability of cloud services a very complex task. In this paper we have identified the challenges involved in portability of cloud apps and analyzed the existing techniques for portability at platform level. In this paper, we propose an approach using Model Driven Engineering to develop SaaS applications like CRM in a cloud-agnostic way. We introduce DSkyL, an eclipse plugin for cloud application development using feature models and domain model analysis, which would support construction, customization, development and deployment of cloud application components across multiple clouds. It also reduces the application development time drastically. This paper aims to sketch the architecture of DSkyL and the major steps involved in the development and deployment process.

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Keywords: CRM, Portability, Cloud configuration files, DSkyL, PaaS, Feature models

1. Introduction

Cloud computing is an emerging computing terminology which offers benefits for users to access their application anytime, anywhere. It also offers certain advantages such as high scalability, reduced IT costs, self-service on demand, and pay-as-you-use price models which has gained the attention of today's IT world. A large number of small and medium businesses are now moving to cloud to reduce their infrastructure and operational cost and also to avail cloud services like elasticity and scalability. The increasing popularity has caused rapid growth in the cloud market and vendors in the cloud market have proliferated in the recent years, including tech giants like Amazon,

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)
doi:10.1016/j.procs.2015.04.063

Google, Microsoft and SalesForce[1]. Each of them promotes its own cloud infrastructure, and hence incompatibility in standards and formats to access the cloud has become a big issue. Such incompatibilities prevent them from being widely accepted. Many organizations have found it difficult to adopt cloud-based solutions, particularly because of the vender lock-in problem [2][3] and the huge investment and effort required to transform non cloud applications to cloud. One of the main obstacles faced by organizations is the lack of a general process to help application developers in selecting the cloud provider and services best suited for their application [4], but also in carefully. However, different cloud application platform offerings are characterized by considerable heterogeneity. Because of incompatibilities, users who develop applications on a specific platform may encounter significant problems when trying to deploy their application in a different environment [5]. Hence, the need for multiple clouds to support same application and be able to work seamlessly i.e. cloud portability, is rising [6].

This paper aims to propose an approach for cloud application portability. DSkyL is a development platform (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 is supported by the different cloud configurations. DSkyL also helps the developers to decrease development effort and time.

The rest of the paper is organized as follows: In section 2 we describe the related work. Section 3 talks about the challenges in portability of cloud applications. In Section 4 we have presented our proposed method, architectural and implementation overview. Section 5 concludes the paper.

2. Related Work

NIST defines portability as the ability of customers to move their data or applications across multiple cloud environments at low cost and minimal disruption [8]. Cloud portability, or the ability to migrate a cloud-deployed asset to a different provider, is a direct benefit of overcoming vendor lock-in [9]. Given the different characteristics of each cloud service model, the concept of portability depends on the service model adopted. According to [10], IaaS portability is the migration of virtual machines. PaaS portability is the migration of code and data [11].

Ensuring portability across cloud providers would eliminate the vendor lock-in problem [12] 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.

2.1. Existing Approaches for Paas Portability

Recently, several initiatives have emerged that define approaches to support application migration to the cloud. A comparative study of different approaches is summarized in this section.

1) Open Cloud Computing Interface (OCCI): OCCI provides set of specifications for cloud tasks like deployment, dynamic scaling and monitoring across different cloud providers. It offers an API which is supported by Eucalyptus, OpenNebula and OpenStack. Hence, OCCI can be classified as a standardized approach for Open Cloud Computing Interface [13].

2) SimpleCloud: SimpleCloud is an API that allows to use storage services independent of cloud platforms. It offers two key services (i) File Storage Service and (ii) Document Storage Service. The File Storage Service allows file operations such as storing, reading, deleting, copying etc. It allows developers to access storage services from Amazon, Microsoft Azure, Rackspace and others, using the same application code. The Document Storage Service provides developer a single API that abstracts the interfaces of all major databases. SimpleCloud can be considered as an intermediary layer for decoupling applications from directly accessing the storage mechanisms of specific platforms [14].

3) mOSAIC: mOSAIC provides an Agnostic, vendor neutral, API at PaaS level and an Open Source Platform, with adapters to most notable Cloud Providers' APIs. It also deals with Cloud Agency for multi Cloud Services

brokering, SLA monitoring and dynamic reconfiguration. mOSAIC also proposes a machine-readable Cloud Ontology. At design-time, using these API developers can create applications that consist of multiple cloud components. A cloud component for example, can be a Java application. At this point the application is not bound to any specific platform. mOSAIC platform decomposes the application into the various cloud components and deploys each one on the cloud platform that provides the best implementation for the cloud component's functionality [15].

4) OASIS TOSCA: TOSCA aims to enable interoperable infrastructure cloud services and description of application, the relationships between parts of the service, and the operational behavior of these services. TOSCA will also make it possible for higher-level operational behavior to be associated with cloud infrastructure management. The TOSCA specification uses TOSCA xml and xs namespace prefixes [16].

5) MODA Clouds: Model Driven approach for Design and Implementing application on multiple cloud allows allow developers to design software systems in a cloud-agnostic way and to be supported by model transformation techniques into the process of instantiating the system into specific, possibly, multiple Clouds [17]. During design, implementation and deployment, the MODACLOUDS Integrated Development Environment (IDE) supports a Cloud-agnostic design of software systems, the semi-automatic translation of design artifacts into code, and the automatic deployment on the targeted Clouds.

6) Openshift: OpenShift is a Platform as a Service offered by Red Hat. Openshift is a platform for developers to build, test, deploy and run cloud applications [18]. By using this developer can focus only in designing and coding, whereas all the infrastructure and middleware management is handled by Openshift. Openshift supports No-Lock-In at PaaS level by providing built-in support for Java, Python, PHP, Perl, Ruby and Node.js.

7) ARTIST - ARTIST proposes an approach that starts with the characterization of application from two points of view; technical and business of the current legacy application and how the company expects those aspects to be in the future to provide a gap analysis. It is then followed by a technical feasibility analysis and business feasibility analysis. Based on this gap analysis using a technical feasibility tool and a business feasibility tool, the migration tasks and their effort are recorded, and it also simulates the impact of the modernized application in the organization [20].

3. Cloud Application Portability Issues

Portability in the cloud can refer to two different but interlinked aspects:

i) Legacy software's modernization aimed at exploiting current cloud-based technologies.
ii) Portability of cloud-ready applications among different cloud platforms and providers.

To proceed with; we have identified the specific points of conflict which arise while an application is ported from one cloud platforms to another. We have identified those aspects of an application which need to be addressed differently in cloud platforms. In this section we discuss the following few potential conflict points: programming languages and frameworks, platform-specific services and platform specific configuration files [19].

a) Programming languages and/or frameworks - Each cloud platform has support for certain language, versions and frameworks. The specific programming languages and frameworks that an application has been built will be a major concern while porting it to another platform. For example, while Google App Engine (GAE) provides support for Python and Java, Amazon supports Java, .NET, PHP, Python and Ruby.

b) Platform specific services –The time taken for application development can be drastically reduced by using API's. 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.

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 [20].

There are two cases:

i) The target platform doesn't provide the full set of services that the application uses. In this case the developer would need to recreate the missing functionality from scratch on the new target platform.

ii) 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) Platform specific configuration files – Cloud platforms require configuration files in order to instruct the hosting environment and execute application unlike traditional software applications. For example Google App Engine uses the "appengine-web.xml" file [20]. 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.

4. Proposed Approach

In this paper we propose an approach which is commonly summarized as "model once, generate anywhere [21]" which emphasis on application deployment as well as migration of the application from one cloud to another. Given this, several research groups are combining model-driven engineering with cloud computing, including ModaClouds and the Advanced Software-Based Service Provisioning and Migration of Legacy Software which is discussed in section 2.

Our solution DSkyL is an eclipse plug-in for modeling, validating and generating platform specific image file for a CRM application. The idea is to provide a default template for the application which can be further refined to add or remove features of the application as well as cloud services according to the developer's or user's choice. Our focus is on reducing the effort and time taken by developers and users in understanding the terminologies used by different CRM and cloud vendors. We concentrate on delivering the applications and services in terms of "features" which do not require any technical training and can be easily understood by anyone.

4.1. Why Feature Models and CRM

End users and developers have different perspectives about the software [22]. User focuses on the problem domain, where system's features are the primary concern. Developer focuses on the solution domain, where life-cycle artifacts are of importance. Hence there arises a major difficulty in understanding the system because of different interest of the user and the developer. A feature is a bundle of system functionality that focuses on the system from the user's perspective. Users generally request new functionality or report defects in existing functionality in terms of features. Developers are expected to translate such feature-oriented requests to life-cycle artifacts.

In DSkyL feature model is represented as a feature tree where nodes represent features and edges represent the "selection" relationships among features. From a feature model, a specific variation of a product can be derived by selecting the desired features based on customer's requirements and feature relationships can be specified in the feature model.

The traditional approach for using CRM requires purchase of databases, server, necessary hardware and software at the customer organization itself. The Maintenance and support costs of these can be very huge every year. The customer organization has to dedicate people to handle all the upgrades as well as backup systems.

The hosted approach allows an organization to host their application in the space on a server owned or leased for use. The customer purchases the software licenses but is not responsible for maintaining the operating system,

database, or a disaster recovery plan. Those aspects would be taken care by the web hosting company. Customizations may or may not be performed by the hosting provider.

The SaaS approach does not require the purchase of server hardware or software. The software, along with maintenance and support and disaster recovery, are provided via a subscription and delivered over the Internet. Software upgrades can be continuous so customers are not forced to plan for and spend time on upgrade efforts. Customizations are performed by the provider at the customer's direction.

Even though Cloud Computing has transformed the way systems are developed, vendor lock-in is a major issue which holds back organizations in using SaaS applications. In this paper we try to address two major concerns on SaaS applications.

i) We offer a PaaS for the developer so that he can customize the CRM application as he wants (customization).
ii) The application that is created using a PaaS will be deployable across multiple cloud platforms (portability).

We present our initial results of our PaaS prototype, built to support feature models and support for multiple clouds. The prototype demonstrates that the developed language, despite being simple, can be used as input to a full code generation process targeting a cloud platform.

4.3. Architecture and Implementation Overview

Organizations and the development community is hesitant to create their systems using certain specific technologies and later being charged with unfair rates for its usage. They are even reluctant to choose a technology which may turn out to be inadequate or inefficient in near future. In order to take advantage of the flexible cloud architecture, the applications have to be specifically developed for the chosen cloud platform. For example, in order to offer great elasticity, Google App Engine – a PaaS provider – imposes a specific programming style and it has its own way to manage data, and thus an application developed specifically for GAE and the data associated with it may not be easily ported to a different PaaS provider. Even if a developer want to host an application in his/her cloud later, additional effort would be required to rebuild the application, redeploy it and migrate all the data. A re-engineering process required to change the cloud provider can be costly and time consuming.

Consider the scenario where you are a developer at an ISV (Independent Software Vendor) that offers CRM application on one of the most popular SaaS platforms available. Now if you want to sell your application to those customers using alternative platforms and if some of those potential customers want to have the application hosted in a different environment; the application have to be re-written to run on those environments and build a new cloud hosting relationship. As an ISV, this would be very expensive. Such a scenario limits the "openness" at the platform level. Platforms which use a proprietary programming language, explicitly tied to a single vendor's implementation will force the customers to use a specific platform thereafter.

Our solution DSkyL targets developers and CRM vendors by providing them a solution that supports the development of Cloud-based CRM software applications which are portable across multiple cloud platforms. We use feature models to drive the development and migration planning process. These feature models are platform-independent that captures the essence of the application to produce a domain specific code (DSL). The deployable file is generated from the cloud configuration file and the source codes corresponding to the added features. This deployable file is a platform independent image and it is specific to each cloud service provider. Porting an application from one cloud platform to another requires transformations with same set of models, which represent the functionalities and different cloud configuration files generated from the SLAs.

DSkyL is an eclipse plug-in for modeling, validating and generating platform specific image file for a CRM application. The idea is to provide a default template for the application which can be further refined to add or remove features according to the vendor's or developer's choice. Whenever the user creates a new project, a

package is created which contains a template feature model with basic functionalities for the selected application type (CRM), source files corresponding to the default feature model, a Model.xml file for representing the features supported a Model.conf file which shows the feature hierarchy and its dependencies. If the user wishes to modify the default features, user has to create new feature in the Model explorer window. Once the feature model is updated, the Model Validator will evaluate the feature dependencies; verify the constraints and impose rules on it. After verifying the feature model; the Model validator will check for the added features and its dependencies. If the designed feature model fails the validation the Model validator will produce an error message. If the feature model passes the validation, the designed model is ready for execution to generate the source files corresponding to every feature. After model validation when the user runs the project, .jak files are generated for all the features in the feature diagram. DSkyL uses AHEAD composer for converting .jak files to corresponding .java files. When user clicks on the run button, DSkyL calls a batch file which executes commands to convert the .jak files to .java files. The generation of .jak files for all the selected features and calling the batch file for running jak2java command occurs simultaneously once the user clicks on the run button. And finally the source code is generated for every feature. The sales order then passes in to the back end system for further processes. So constraints like sales order is generated only once the quotation is won also verified by our tool.

The following snapshots illustrate the steps involved in developing feature based application using the prototype DSkyL tool and the deployment of a selected feature in IBM Bluemix Cloud.

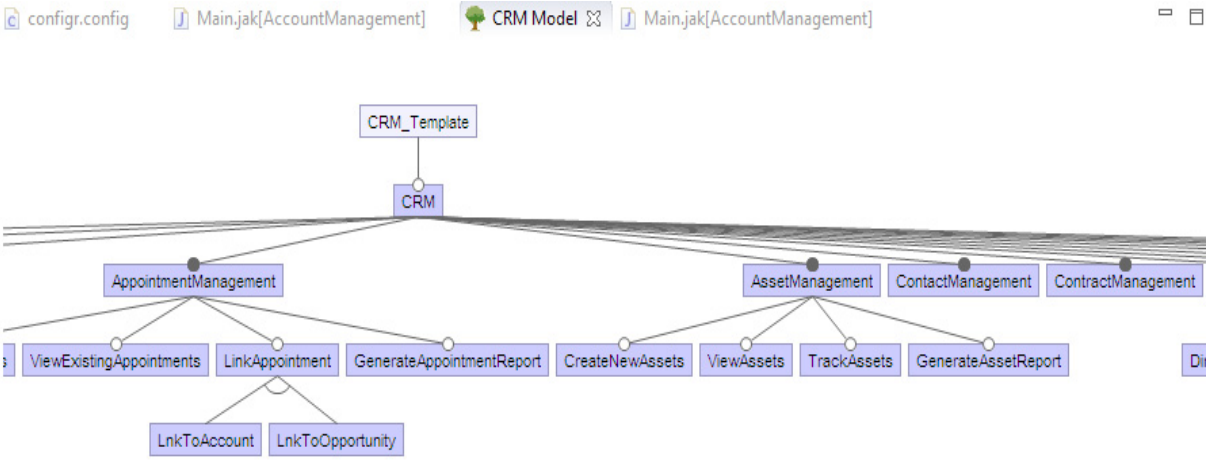

Fig.1 : The default feature model created in eclipse workspace after creation of the project.

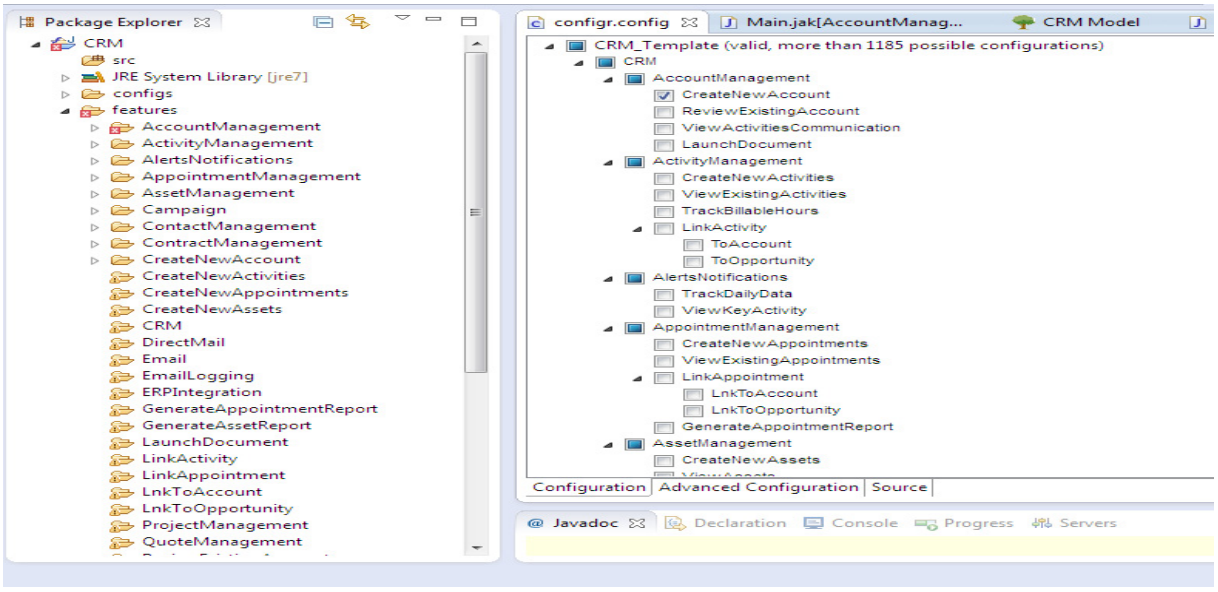

Fig. 2: Developers can select the required features and confirm them. Corresponding codes will be generated for selected features

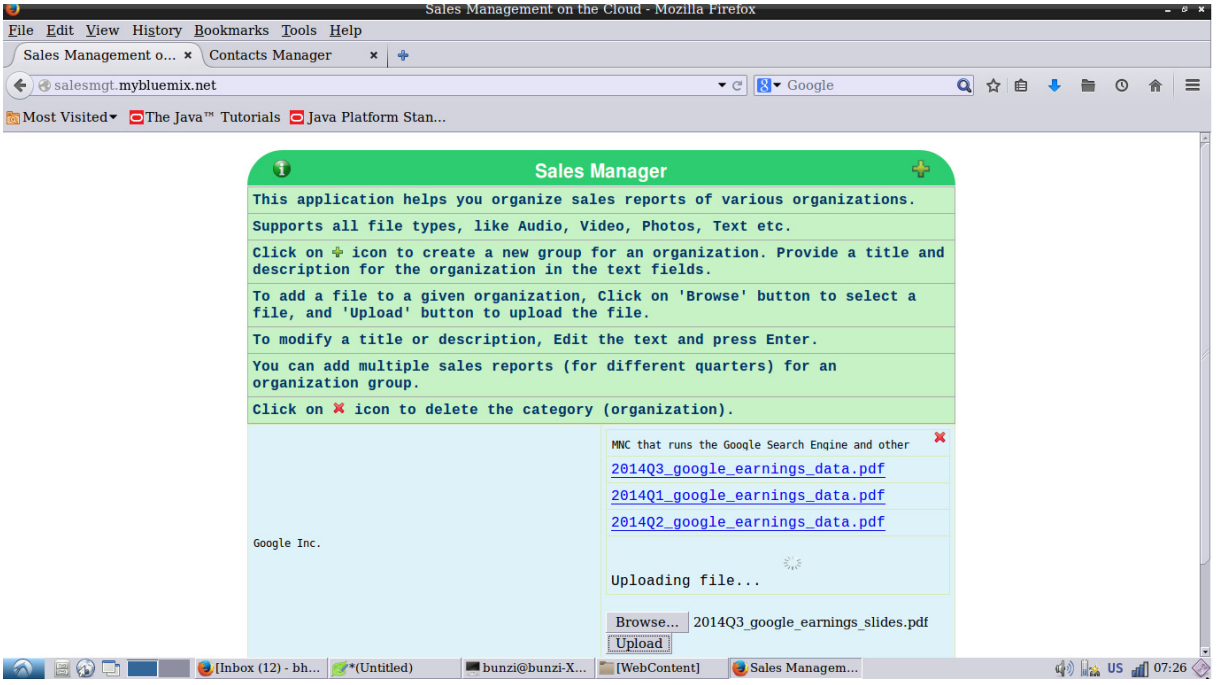

Fig. 3: Deployment of selected feature in IBM Bluemix cloud

The final image file created by DSkyL will be a combination of the model and this configuration file. This is specific for every cloud provider. Each PaaS provider offers a special flavor in its design. DSkyL framework offers a solution for the development of portable and Cloud provider independent applications. It enables the developer to build Cloud applications in a very flexible way, being completely independent from the Cloud providers that offer the resources. DSkyL uses the concept of wrappers to allow users to access multiple Cloud resources. Each Cloud

service provider has his own architecture for storage or communication and hence wrappers are created so that it becomes portable across multiple platforms. A contract will grant user's requirements and the Resource Manager will assign physical resources on the basis of the contract.

5. Conclusions

In recent days the vendor lock-in problem has evolved as a major hindrance for cloud computing being widely adopted. It is because users/organizations may opt for different cloud providers over a period of time for various reasons like optimal choice on expenses and resources, contract termination or some legal issues. In order to solve this problem, this paper presents a model-driven approach for cloud portability. The cloud technologies and MDE, together, can benefit the users by providing better productivity, improved maintenance and reuse.

References

[1] Zhizhong Zhang, Chuan Wu, David W.L. Cheung, A survey on cloud interoperability: taxonomies, standards, and practice, ACM SIGMETRICS Performance Evaluation Review, Volume 40 Issue 4, March 2013

[2] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010, pp. 27–33.

[3] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K.Tarabanis, "Towards a Reference Architecture for Semantically Interoperable Clouds," 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 2010, pp.143–150.

[4] S. b Yangui and S. Tata, "PaaS elements for hosting service-based applications," in CLOSER 2012, 2012, pp. 476–479.

[5] V. Nelson and V. Uma, "Semantic based Resource Provisioning and scheduling in inter-cloud environment," in International Conference on Recent Trends in Information Technology, 2012, pp. 250–254.

[6] Sampaio and N. Mendonça, "Uni4Cloud," in 2nd Intl. workshop on Software engineering for cloud computing, 2011, pp. 15–21.

[7] S. Charrington, "Don't Pass on PaaS in 2010," (ebizo), [online] 2010, http://www.ebizq.net/topics/cloud_computing/features/12279.html?page=2, (Accessed: 2014)

[8] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger and Dawn Leaf, ―NIST Cloud Computing Reference Architecture, National Institute of Standards and Technology, Gaithersburg, 2011

[9] P. Mell and T. Grance, "NIST Definition of Cloud Computing," Gaithersburg, 2011.

[10] Govindarajan and Lakshmanan, "Overview of Cloud Standards," in Cloud Computing, London: Springer London, 2010, pp. 77–89.

[11] Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar, "Winds of Change: From Vendor Lock-In to the Meta Cloud," IEEE Internet Computing, vol. 17, no. 1, pp. 69–73, Jan. 2013.

[12] Neal Leavitt, ―Is Cloud Computing Really Ready for Prime Time, Computer, vol. 42, no. 1, pp. 15–20, 2009.

[13] N. Loutas, E. Kamateri, and K. Tarabanis, ―A Semantic Interoperability Framework for Cloud Platform as a Service, in 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), Athens, 2011, pp. 280 ‐ 287.

[14] Fotis Gonidis, Iraklis Paraskakis, Dimitrios Kourtesis, Addressing the Challenge of Application Portability in Cloud Platforms, BCI13

[15] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, ―Portable Cloud applications—from theory to practice, Future Generation Computer Systems, 2012.

[16] Magdalena Kostoska, Marjan Gusev, Sasko Ristov, A New Cloud Services Portability Platform, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013

[17] Danilo Ardagna, Elisabetta Di, Giuliano Casale, Dana Petcu , Parastoo Mohagheghi, S´ebastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, Cosmin-Septimiu Nechifor, Craig Sheridan, MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds, MiSE-2012.

[18] Redhat: http://www.redhat.com/developers/openshift/

[19] Fotis Gonidis, Iraklis Paraskakis, Anthony J. H. Simons, Dimitrios Kourtesis, Cloud Application Portability: An Initial View, Balkan Conference in Informatics, BCI '13, Thessaloniki, Greece, September 19-21, 2013

[20] Aparna Vijaya, Pritam Dash, Neelanarayanan V, Migration of Enterprise Software Application to Multiple Clouds: A feature based approach, LNCS Vol 3 No 2 May 2015

[21] Jean Bezivin, Model Engineering-From principles to platforms, March 2005.

[22] C. Reid Turner, Er L. Wolf, Luigi Lavazza, Alfonso Fuggetta, A Conceptual Basis for Feature Engineering, The Journal of Systems and Software 49,1999.