I.J. Education and Management Engineering, **2015, 4, 27-35**
Published Online September 2015 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijeme.2015.04.04

Available online at http://www.mecs-press.net/ijeme

A Model Driven Framework for Portable Cloud Services: Proof of Concept Implementation

Aparna Vijaya [a], Neelanarayanan V [a]

[a] *Vellore Institute of Technology of Technology, Chennai, 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 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 process.

Index Terms: Portability, vendor neutral, model driven.

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 benefits like elasticity and scalability. The increasing popularity has caused rapid increase in the number of cloud vendors in the market. Each of them promotes its own cloud infrastructure, and hence incompatibility in standards and formats to access the cloud has become a major issue. Such incompatibilities prevent them from being widely accepted. Many organizations have found it difficult to

* Corresponding author
Email:

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 understanding in selecting the cloud provider and services best suited for their application [4]. 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 [5]. Hence, the need for multiple clouds to support same application and be able to work seamlessly i.e. cloud portability, is rising [6].

Standardization could be a simple way to achieve cloud portability, however, it may take years for the standards to be fully agreed upon and adopted, if ever. Hence, effort has been made from researchers for developing technologies to enable portability among clouds, from both the cloud provider's and user's perspectives.

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 follow: In section 2 we describe the related work. Section 3 talks about our proposed method, architectural and implementation overview. And Section 5 concludes the paper.

2. Background

In this section the need for deploying cloud applications over heterogeneous cloud providers and few existing solutions are discussed.

2.1. Need

Assume a startup company working on a mobile application development. They have developed the application with Slim REST API framework and pushed it into the HP Horizon Cloud as they were giving a free usage for limited use for its users. As months pass by the customer base grew gradually and now there is a need to step into other domains like Windows and iOS. Since the organization does not have a huge developer division, the cost for hiring separate platform programmers will incur more cost than the revenue it generated. This is a scenario where the need for application portability comes in.

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.

Hence a platform independent image can be beneficial in four scenarios:

i) When application development and hosting is not particular to a cloud provider.
ii) When a cloud service provider aims to improve their services by providing new application development APIs and hosting methodologies.
iii) When a cloud service provider aims to scale their services (SaaS) by offering the services of a new cloud provider's resources (IaaS).
iv) When a user needs more processing power for his application, he might want to host the application in multiple clouds. The equations are an exception to the prescribed specifications of this template.

2.2. Existing Literature

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, OpenNabula and OpenStack. Hence, OCCI can be classified as a standardized approach for Open Cloud Computing Interface [11].

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 [12].

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 this 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. Then, at runtime, the 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. The selection of the concrete cloud services to be used is automated and performed by the mOSAIC platform. Therefore developers can focus on developing their applications in a platform-neutral manner, and later on, they can decide on which cloud provider they wish to deploy them [13]. The mOSAIC API acts as an intermediary layer between the developers and the actual cloud platforms, and developers do not have to use proprietary APIs of the target could platforms.

4) OASIS TOSCA: The OASIS TOSCA works to enhance the portability of cloud applications and services. 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 [14]. The specification defines a meta-model for defining the structure of an IT service and its management. The Topology Template defines the structure of a service. Plans define the process models that are used to create, terminate and manage a service.

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 [15]. 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. The run-time layer offered by MODACLOUDS: (i) enables system operators to oversee the execution of the system on multiple Clouds; (ii) automatically triggers some adaptation actions (e.g., migrate some system components from a IaaS to another offering better performance at that time); and (iii) provides run-time information to the design-time environment (the IDE) that can inform the software system evolution process.

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 [16]. By using this developer can focus only in designing and coding, whereas all the infrastructure and middleware management is handled by Openshift. Steps for developers to use Openshift: i) Create an "Application" in OpenShift (using command-line or the IDE). ii)

Code the application (in IDEs like TextMate, Eclipse, Visual Studio etc). iii) Push the application code to OpenShift (again, with the command-line or the IDE). Openshift supports No-Lock-In at PaaS level by providing built-in support for Java, Python, PHP, Perl, Ruby and Node.js. In addition, OpenShift is extensible with a customizable cartridge functionality that allows enterprising developers to add any other language they wish. In addition to this flexible, OpenShift supports many of the popular frameworks such as Spring, Rails, Play that make developer's life easier.

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 [17].

A comparative study performed on these methodologies is summarized as follows:

Table 1. Comparison of cloud portability approaches

Approach Name	Languages supported	Data Support	OS	Clouds Tested	Vendor Independent?	Methodology adopted
OCCI	Java, Ruby, erlang	AWS	Cross platform	OpenNebula, mnesia	Yes	Cloud-specific standards
SimpleCloud	PHP	Amazon S3 and Nirvanix IMFS	Cross platform	Zend Cloud	No	API
mOSAIC	Java, Python, erlang, node.js	Riak, CouchDB, MemcachDB, Redis, MySQL, Amazon S3, HDFS	Linux	Amazon EC2, OpenNabula, Eucalyptus,	Yes	Multiagent Systems
OASIS TOSCA	Java, PHP	MySQL	Linux	OpenStack	Yes	Cloud-specific standards
MODA Clouds	Java, Python, erlang, node.js	Riak, CouchDB, MemcachDB, Redis, MySQL, Amazon S3, HDFS	Linux	Eucalyptus	Yes	Model-driven application engineering
Openshift	Java, Python, perl, Ruby, PHP, .NET	MongoDB, MySQL, PostgreSQL, Microsoft SQL Server	Linux	-	Yes	Hybrid Platform as a Service
Artist	Java	MySQL	Cross platform	-	Yes	-
Docker	Java, Python, Ruby, node.js	MongoDB, Postgre SQL	Cross platform	Rackspace, Joyent, Azure, Google Cloud	Yes	Container & packaging approach

3. DSkyL – Proof of Concept

3.1. Overview

Traditional cloud computing providers enable developers to program and deploy applications in the cloud by using platform as a service (PaaS). PaaS based cloud computing allows cloud customers to concentrate on

application development and maintenance whereas the underlying plat-form is managed by a cloud provider. Despite the benefits of such an approach, since the PaaS is specific for each vendor; this service model usually comes with a high level of vendor lock in. Whenever the developer decides to change its cloud provider there will be a major application rewrite, specific for the new PaaS. This section details the initial results of a work whose goal is to build a PaaS which will create applications which are platform neutral.

It is always recommended to keep pieces of the applications as small as possible. If you're building out a giant application that's going to do multiple things, architect in such way that it is broken down into multiple smaller applications. Different cloud services run on different hardware, some with better performance than others. So if the company decides to move to another cloud service with the application, building it in small chunks makes it easier to run on multiple machines.

We have used the concept of feature models to decompose the application to small chunks. 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 configuration files. Portability is achieved by using the concept of containers. Containers will include everything that is needed for an application to run. A container will include all the dependencies associated for running the application.

3.2. Architecture

The proposed method allows the application to be ported based on: (i) an architecture model of the application (ii) deployment model or topology of the application and (iii) service level agreements for application deployment in cloud.

There is an abstraction between the application and the underlying cloud platform. Hence end users can quickly build and deploy applications without worrying about the underlying environment. This reduces the concern of missing dependencies due to differences in the underlying operating system. A high level architecture specifying where DSkyL fits in is specified below:

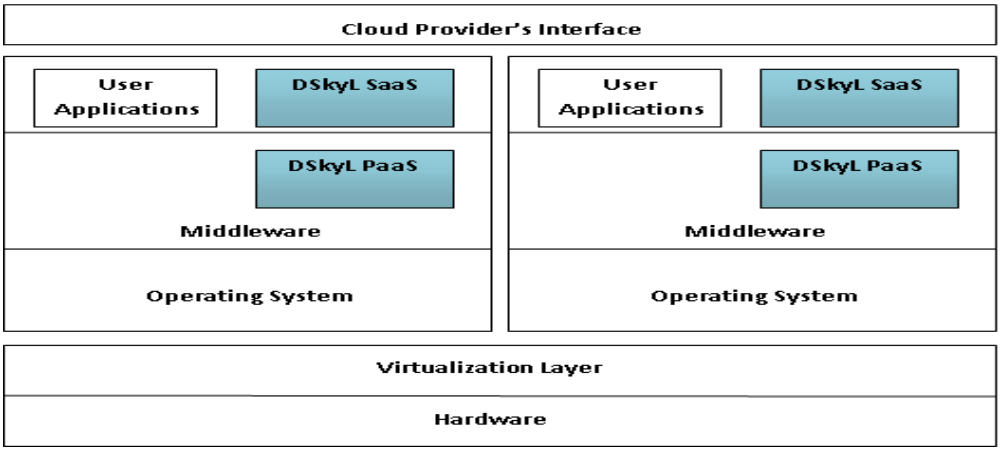

Fig.1. High Level Architecture of DSkyL

3.3. Modeling Notations

We have used the concept of nodes and relationship type for specifying the application components and their topology. The service level agreements or constraints can also be imposed on the model by mentioning them as a rule.

End users and developers have different perspectives about the software [20]. 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.

To create an application, the first step is to create a new architecture template in which the components of the application are modeled. Every component is specified as a feature while describing the architecture of the application. Feature is a set of modules that implement a subset of system functionality. They are a user-centered view of a system's functionality and is an aspect in the problem domain. Every Feature Model has a root feature beneath which other optional or mandatory features are modeled. The dependencies between the features can also specified using relations. The relationship currently supported by the tool is "compose of", "refines", "conflicts", "requires" and "mutually exclusive". Features for an application can be either optional or mandatory.

Once the application is modeled; the deployment topology for the application is defined using deployment template. The deployment topology can be specified using nodes where every node can be either a server or application or database and so on. The features are related to the nodes using relationships. Also the nodes can be related to one another using relationships. Three relationship types are by default implemented in the tool. They are "deployed on", "hosted on" or" installed on". User can create new relationship type according to the need.

The tool offers all available features and nodes in a palette in its template. From there, the user drags the desired features or nodes and drops it into the editing area. Selecting one relationship type creates a relationship that has to be connected to the desired source and target.

3.4. Container Overview

TOSCA is currently advanced in its specification for orchestration which makes it an ideal candidate for becoming the standard blueprint definition for containers. The fact that TOSCA is backed by a standards body (OASIS) makes it a great platform for defining a standard container orchestration specification that is portable across various cloud environments and container providers. We have used the Cloud Service Archive (CSAR) proposed by OASIS TOSCA. It is a container file in the ZIP file format. It includes the cloud application architecture and its deployment template, all artifacts like operations defined by every node type and artifacts required to execute the application like associated libraries, deployment descriptors and other configuration files.

It represents metadata of the other files in the CSAR. These metadata are given in the format of name/value pairs. These name/value pairs (<*name*>*:* <*value*>) are organized in blocks. Each block provides metadata of a certain artifact of the CSAR. The structure of the manifest file is as follows:

Manifest-Version: x.x
CSAR-Version: x.y
Created-By: test
Entry-Service-Template: file name that is the entry point for the cloud application.

3.5. Implementation

DSkyL is as an Eclipse plug-in and it adheres to OSGi architecture for plug-in development. It has a Modeling Tool Editor with application modeler template and deployment template. These templates facilitate the creation of application features and deployment topology.

Sample of application template and deployment topology template is shown below:

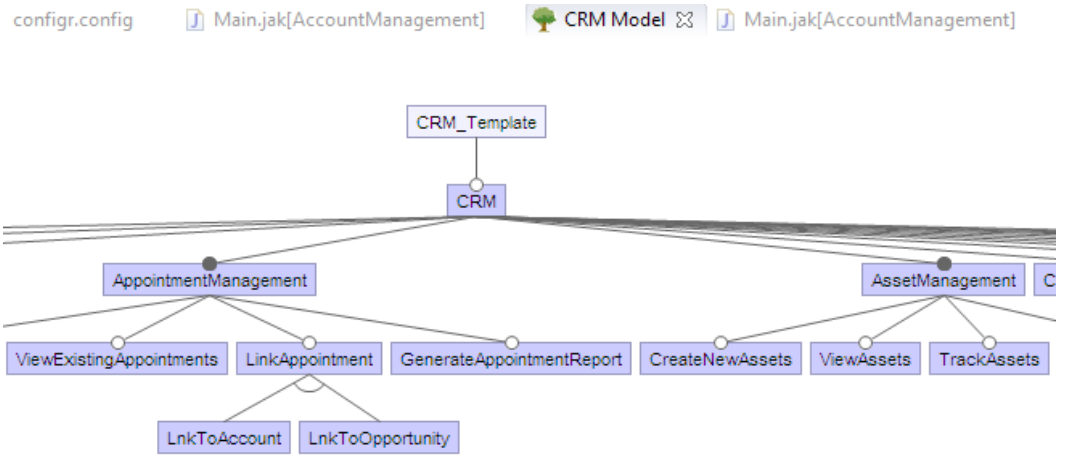

Fig.2. Application Topology

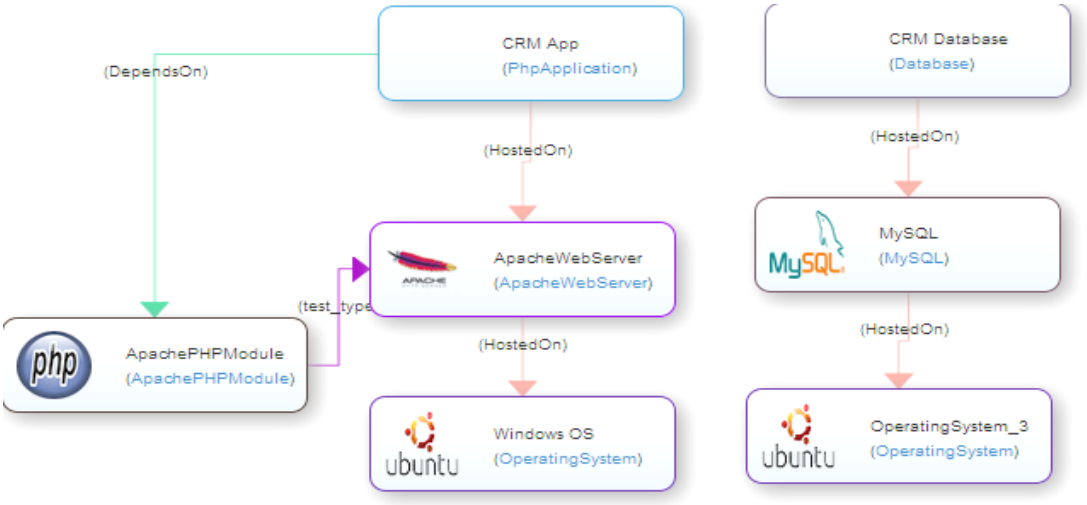

Fig.3. Deployment Topology

4. Conclusions and Future Work

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. The lack of standard approaches for portability between cloud providers causes the problem. It might take years for providers to agree upon common standards. 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. The proposed approach is still under development. Only a prototype of the tool has been implemented. Currently data portability has not been implemented and the evaluation of the tool is not been carried out.

References

[1] Aparna Vijaya, Neelanarayanan V, "Framework for Platform Agnostic Enterprise Application Development Supporting Multiple Clouds", Proc. Symp of BigData and Cloud computing Challenges - Elsevier Procedia Computer Science (ISBCC 2015, March) (Pending publication).

[2] Aparna Vijaya, Neelanarayanan V "Platform Agnostic Application development", Computational Intelligence for Big Data Analysis: Frontier Advances and Applications, Springer-Verlag, Heidelberg, Germany, Series - 'Studies in Adaption, Learning, and Optimization'.

[3] Aparna Vijaya, Pritam Dash, Neelanarayanan V, "Migration of Legacy Enterprise Applications to Multiple Clouds: A Feature based approach". Lecture Notes on Software Engineering (LNSE, ISSN: 2301-3559, DOI: 10.7763/LNSE) Journal.

[4] Aparna Vijaya, Neelanarayanan, " Survey on Decision Framework for Migration to Cloud", Proc. of International Conference on Mathematical Computer Engineering (ICMCE 2013, November).

[5] "Cloud Computing Portability and Interoperability: Portability and InteroperabilityInterfaces", http://www.opengroup.org/cloud/cloud/cloud_iop/interfaces.htm. (2015).

[6] 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.

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

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

[9] 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.

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

[11] 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.

[12] Fotis Gonidis, Iraklis Paraskakis, Dimitrios Kourtesis, Addressing the Challenge of Application Portability in Cloud Platforms, BCI-13.

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

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

[15] 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.

[16] Redhat: http://www.redhat.com/developers/openshift/ (2015).

[17] 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.

[18] "Docker User Guide", https://docs.docker.com/userguide (2015).

[19] "Portable cloud apps require standards, vendor research", http://searchcloudapplications.techtarget.com/tip/Portable-cloud-apps-require-standards-vendor-research

(2015).
[20] C. Reid Turner, Er L. Wolf, Luigi Lavazza, Alfonso Fuggetta, A Conceptual Basis for Feature Engineering, The Journal of Systems and Software 49, 1999.

Author(s) Profiles

Aparna Vijaya is Currently pursuing PhD in Software Engineering. Masters of Science (MS) in Computer Software Engineering from Mälardalen University, Sweden. Bachelor of Technology (B.Tech.), Information Technology from Amrita Vishwa Vidyapeetham, India. Her major area of work is Software Engineering and Model Driven Development. She has 5 years industrial experience with Tata Consultancy Services.

Neelanarayanan V is Associate Professor at Vellore Institute of Technology, Chennai Campus. He has completed his PhD from IT University of Copenhagen, Denmark. He has 7 years of R & D experience from CDAC, India and 6 years of teaching experience.

How to cite this paper: Aparna Vijaya, Neelanarayanan V,"A Model Driven Framework for Portable Cloud Services: Proof of Concept Implementation", IJEME, vol.5, no.4, pp.27-35, 2015.DOI: 10.5815/ijeme.2015.04.04