Migration of Enterprise Software Applications to Multiple Clouds: A Feature Based Approach

Aparna Vijaya, Pritam Dash, and Neelanarayanan V.

Abstract—Cloud computing is a growing buzz in today’s IT world. Many organizations are now adopting cloud computing techniques in industrial applications. As a result of the growing interest towards hosting applications in cloud there are a number of cloud service providers in the market. New products are launched continuously to make cloud hosting simple and feasible. But there are some challenges that prevent cloud platforms from being widely adopted. One of the major challenges in adopting cloud computing is the portability of applications across various clouds. Cloud based applications are normally provided as services offered by different cloud service providers. The different implementations of cloud computing services developed by each cloud provider resulted in incompatibility of APIs which the cloud application developers find it difficult to understand and implement. This is called vendor lock-in and it increases the effort in application development. In many cases there is a need for supporting multiple clouds since one cloud provider may not always cater to the needs of an organization. And they might require a different cloud service for their applications. This paper aims to systematically analyze the different existing approaches on cloud migration to analyze the gap and report an approach based on feature models. DSkyL is an eclipse plugin which is in development phase which uses feature models to generate platform independent image files. The paper gives an overview about the architecture of DSkyL.

Index Terms—CRM / ERP, cloudlets, cloud configuration files, DSkyL, feature models, platform independent image.

I. INTRODUCTION

Cloud computing is an emerging computing terminology which offers benefits for users to access their application anytime, anywhere. It also offers 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 world. Many organizations have found it difficult to adopt cloud-based solutions, particularly because of the vendor lock-in problem and migration related issues of their existing legacy applications to this new cloud environment.

One of the main obstacles faced by those organizations is the lack of a general process to help application developers not only in selecting the cloud models and services best suited for their application, but also in carefully assessing the feasibility, various risks and benefits involved. The growing interest towards cloud application platforms has resulted in a large number of platform offerings available on the market for use. However since these platform support only a specific set of configurations, portability becomes a major concern. 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 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.

A. Need for Supporting Multiple Clouds

A large number of small and medium businesses are now moving to cloud to reduce their infrastructure and operational cost and also to increase the scalability. It takes a longer time for them to implement their application from scratch and a lot of effort when supporting the necessary changes that needs to be done for supporting the different cloud vendors.

Unfortunately every cloud platform is different from another. That is they are heterogeneous. Because of these 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. 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.

II. TECHNIQUES FOR MODERNIZATION OF APPLICATION TO CLOUD

Many organizations write their software from scratch specifically for hosting in cloud, while others want to keep their existing application software and run it on a cloud platform. Migration across different cloud is not automatic and the amount of effort required could be significantly high since the architecture of the application might not always support the different cloud providers. There are often differences in various infrastructures, the programming models and the libraries available in traditional and cloud model.

Like any software development project, migration projects require a lot of planning and proper process to follow. Normally migration approaches where one needs to completely rewrite the application or development of a
completely new piece of software is less likely to be chosen due to the cost and time involved. While a cloud migration can present numerous challenges and raise security concerns, cloud hosting enables an organization to reduce capital expenditures and operating costs. Its high availability, multi-tenancy and effective resource allocations are the other key features which most organizations look forward for.

Some of the approaches for modernization of applications are [1]:

1) Wrapping is building a new interface called wrapper, to act as an interpreter between legacy system and target system. Minimum changes are applied to legacy component which implies less cost and lower risk. But wrapping cannot be applied straight-forwardly when the functionality of legacy components is not always suitable for the requirements of target system. Further, the lower level structure, such as platform, language, and software architecture, is unchanged in wrapping. It would not result in long-term benefits and improving the maintainability.

2) Migration switches legacy components to current environment and retains the original data and functionality. It offers benefits such as higher flexibility and ease of maintenance.

3) Maintenance is continuously maintaining the legacy components to meet incoming requirements. The degree of change is less than migration. But the maintainability of system is gradually decreasing. Hence, the cost of maintenance is becoming more and more expensive.

4) Redevelopment completely throws away the legacy components and develops new components to meet the requirement of target system. It tends to a high level risk of failure and a higher cost of redevelopment.

5) Replace component(s) with Cloud supported components. Here one or more components are replaced by Cloud supported components / services. Using Google App Engine instead of a local MySQL database is an example of this migration type.

6) Cloudify is the complete migration of the application. The application functionality is implemented as a composition of services running on the Cloud. Cloudification requires the migration of data and business logic to the Cloud.

A. Existing Approaches

1) SMART — Service-Oriented Migration and Reuse Technique (SMART), proposed by SEI at CMU. It helps organizations analyze legacy systems to determine whether their functionality can be reasonably exposed as services. SMART gathers a wide range of information about legacy components, the target environment, and potential services to produce a service migration strategy for the organization. This approach consists of establishing the context of migration, by discussions with developers using the Services Migration Interview Guide. Once the migration system is feasible, the authors of this approach propose to select from the initial list the candidates’ services and their inputs and outputs. It is platform/vendor/tool independent [2].

2) 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 [3].

3) Using Mash Ups — The mashup migration strategy has six steps. The first two activities are the modeling of the target enterprise business (MODEL) and the analysis of legacy systems and infrastructure (ANALYZE). These activities lead to two main steps: (MAP & IDENTIFY) maps model requirements to legacy components and services identifications; and (DESIGN) models mashup server architecture with Domain Specific Kits (DSKs), which abstracts legacy components. Both the mapping and architecture design activities might cause a loopback to MODEL and ANALYZE activities to re-reconsider some of the decisions. As a result of these major activities, target system service dependency graph has been constructed and mashup architecture will be designed. Defining the Service Level Agreements (SLAs), including non-functional and contextual properties, is the next step (DEFINE) that will be followed by implementation and deployment activities (IMPLEMENT & DEPLOY) [4].

4) Service Oriented Modeling and Architecture (SOMA) — SOMA is an iterative and incremental method on how to plan and implement migration which was developed by IBM. SOMA is structured into seven main phases.

Business Modeling describes the customer’s business. Solution Management adapts the SOMA method to the project needs. The Service Identification phase has aims at finding appropriate services. Three different activities are accomplished to identify service candidates. One of these methods, called Existing Asset Analysis, analyzes existing legacy systems in order to identify services. During Service Specification, services, components and communications are formally defined. Service Realization describes the components in more detail. Parts of the legacy systems are analyzed deeply to determine if they can be used as service implementation. According to SOMA, appropriate legacy functions can be migrated into service implementations in two ways:

- Transformation - converts the legacy function into a new technology which can be implemented as service component directly.
- Wrapping - encapsulates a legacy function with an interface without changing it.

During Service Implementation the whole system is implemented. New components are coded, existing legacy functions are transformed, wrappers are written and the services are composed. In the Service Deployment phase the system is deployed to the customer [5].

5) Six Phased Approach — It includes phases such as Legacy system understanding (LSU) in which knowledge about the legacy applications, source code characteristics, identifying dependencies and legacy system architecture is obtained. Target system
understanding (TSU) phase facilitates the representation of the desired architecture. TSU represents two aspects of the target architecture: (i) the functional aspect, and (ii) the technical aspect. Migration Feasibility Determination checks the feasibility of the migration from technical, economical and organizational perspectives. Candidate Service Identification is categorized into top-down and bottom-up approaches. In the top-down approach, initially a business process is modeled based on the requirements and then the process is subdivided into sub-processes until these can be mapped to legacy functions. The bottom-up approach utilizes the legacy code to identify services using various techniques such as information retrieval, concept analysis, and business rules recovery and source code visualization.

The implementation phase is related to the execution of the migration of the legacy applications. Deployment and provisioning phase tests to determine if the expected functionality is exposed correctly as a service when legacy application is migrated as a service [6].

### TABLE I: COMPARISON OF EXISTING CLOUD MIGRATION APPROACHES

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Modernization Approach</th>
<th>Automated tools</th>
<th>Vendor or tool independent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMART</td>
<td>Migration – Assessment method</td>
<td>No (process based)</td>
<td>Yes</td>
</tr>
<tr>
<td>ARTIST</td>
<td>Migration – Assessment method</td>
<td>No (process based)</td>
<td>Yes</td>
</tr>
<tr>
<td>Using Mash Up</td>
<td>Migration</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SOMA</td>
<td>Wrapping / Transformation</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Six phased</td>
<td>Migration</td>
<td>No (process based)</td>
<td>Yes</td>
</tr>
<tr>
<td>Oracle</td>
<td>Migration</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cloud step</td>
<td>Migration</td>
<td>No (process based)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

6) Oracles Approach — The various phases involved in migration are: Assessment — includes collecting information related to project management, potential cost of the migration, migration approaches, and tools to use. Analysis and design phase-determining the implementation details on the target environment. Migration tasks involve data and application migration. Testing in a migration project usually comprises tasks such as data verification, testing of migrated business logic testing of application interaction with the new platforms and scripts. Optimization phase addresses the issues in performance, hardware configuration, software installation and configuration, initial data loading and facilitating backup and recovery options. Post-Production supports troubleshooting any issues that may come up immediately after the new environment goes live. [7]

7) Cloud Step — It is a step by step process aimed at supporting organizations and application developers in making cloud selection and migration decisions. The process relies on the creation of template-based profiles describing key characteristics of the organization, its target legacy application and one or more candidate cloud providers. These profiles are then cross-analyzed to identify and possibly resolve critical constraints (either technical or non-technical) that may affect application migration to the cloud [8].

A comparative study of the various approaches is summarized in Table I.

### III. 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 which arises when attempting to deploy an application to multiple cloud 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 few potential conflict points: programming languages and frameworks, platform-specific services and platform specific configuration files [9].

#### A. Programming Languages and/or Frameworks

The specific programming languages and frameworks that an application has been built will be a major concern 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, amazon uses DotNet.

#### 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. Each platform provider may offer a wider or smaller range of such specific 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. 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.

3) 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 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.

IV. PROPOSED APPROACH

Organizations and the development community are hesitant to create their systems using one or two 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 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 for it as well as 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.

Model-driven engineering (MDE) is a unification of initiatives that aims to improve software development by employing high-level domain models in the design, implementation, and testing of software systems. MDE provides benefits in terms of productivity, ease of communication and software maintenance, allowing software engineers to work in a higher abstraction level.

In our proposed approach, we extend MDE by incorporating domain models which facilitates migration to multiple clouds. Each PaaS provider offers a special flavor in its design. Portability is possible only between a small numbers of PaaS’s. But the number of platforms available has been tremendously increasing. DSkyL framework offers a solution for the development of interoperable, portable and Cloud-provider independent Cloud applications. It enable developers to build Cloud applications in a very flexible way, being completely independent from the Cloud providers that offer the resources. We provide a method for managing the SLA using which the Infrastructure as a Service which includes managing resources like virtual machines and storage required by the application and quality of the services offered are agreed upon. In DSkyL the functionalities supported by the application and cloud resources are modeled in terms of the functionalities offered, and not according to how they work.

Our solution DSkyL targets developers and CRM/ERP vendors by providing them a solution that supports the development of Cloud-based CRM/ ERP software applications which are portable across multiple cloud platforms. As a future enhancement we will provide support for legacy system migration to 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. The deployable file is generated from the cloud configuration file, the final DSL code 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 a eclipse plug-in for modeling, validating and generating platform specific image file for a CRM / ERP application which is presented in Fig. 1. 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. The screen shots for this are shown in Fig. 2 and Fig. 3.

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/ERP), 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.
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. It will also show suggestions to correct the design. 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. The source files are generated for every feature.

Once the model verification and source code generation is done, the user then defines the SLAs for the cloud provider he has decided upon. The SLA will consist of VM properties (size of RAM, No of processors, CPU processing speed etc), VM allocation rules and Couldlet’s properties. SLA’s are then converted to XML configuration files. 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.

Considering a scenario from ERP; there are two teams working on different modules of a project. The teams are led by two different project leads but they work under the same project manager. In this case we would have three levels of users 1) Project Manager 2) Project Lead 3) Developer. Each of them will have different set of priorities and specialized set of functionalities that they can avail. User at level 1 can avail features available to level 3 users, but the reverse is not supported. The project management features like project planning, resource planning, costing, billing and activity management cannot be defined at level 3 with features such as project deliverables, project schedule, work breakdown structure and payroll. The level 3 user (the developers) should not have access to project management functionalities.
If the ERP developers choose to add the above mention functionalities at the same level without defining proper dependencies and access priorities, the Model Validator will display an error message with the error description. It will also provide suggestions to correct and refine the feature model.

Considering the sales module of the CRM software are focused on helping the Sales team to execute and manage the pre-sales process in an organized manner. All the relevant data of a prospective sales deal like customer, products interested in, expected budget, expected closing date etc are available in one place. Opportunity Management of the CRM system helps the sales representative to carry out activities such as tracking opportunities, including percent chance of close of deal, RFP received, quotation sent and finally whether the deal is won or lost. When opportunity management reaches a “Quotation phase”, a quotation is generated which if “won” gets converted into a sales order. 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 major benefit of DSkyL is that the CRM/ERP vendors do not have to worry about the vendor lock-in problem in cloud hosting that is expected when they use a specific cloud provider’s development platform. If the application vendor plans to update the features or change the cloud service provider, they can edit the feature model and run it to get DSL and source code, add SLAs corresponding to the new cloud service provider and generate the deployable file. This way the application vendors do not have to re-engineer the entire application. Moreover there is not much need for very good technical expertise for creating the application and hosting it in cloud since our tools does most of the works in clicks.

Limitations

DSkyL supports its own set of feature models; it does not allow users to import models created using any other tool.

V. CONCLUSION

The lack of standard approaches for portability between cloud providers causes the vendor lock-in problem. This problem is hindering the cloud adoption, because users may opt for different cloud providers because a certain 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. The prototype for the proposed approach is still under development. Initially, the focus will be only with few open source cloud providers. As a future work, DSkyL will be refined to support migration of existing CRM/ ERP applications to multiple clouds both open source and commercial.

REFERENCES


Aparna Vijaya is currently pursuing her PhD in software engineering and also is working as an assistant professor in Vellore Institute of Technology, Chennai Campus. She received her masters of science (MS) in computer software engineering from Mälardalen University, Sweden, and bachelor of technology (B.Tech.), information technology from Amrita Vishwa Vidayapeetham, India.

Her major area of work is software engineering and model driven development. She has 5 years industrial experience with Tata Consultancy Services. She pursued her MS at Mälardalen University and Software Engineering Institute, 2008. She received her masters of science (MS) in computer software engineering from Mälardalen University, Sweden, and bachelor of technology (B.Tech.), information technology from Amrita Vishwa Vidayapeetham, India.

Pritam Dash is pursuing his masters of science in software engineering (integrated) at Vellore Institute of Technology, Chennai Campus India. Currently he is in third year of the course.

Neelanarayan V. is an 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.

106