Migrating Legacy Applications to the Service Cloud

Conference Paper · October 2009

CITATIONS
26

READS
3,267

4 authors, including:

Weiqing Zhang
Telenor Research
12 PUBLICATIONS **139** CITATIONS

SEE PROFILE

Dumitru Roman
SINTEF AS / University of Oslo
255 PUBLICATIONS **4,358** CITATIONS

SEE PROFILE

Migrating Legacy Applications to the Service Cloud

Weiqing.Zhang, Arne.J.Berre, Dumitru.Roman, Hans.Aage.Huru,

[1]SINTEF, Pb. 124 Blindern, NO-0314 Oslo, Norway
{ davidzhang.job@gmail.com, Arne.J.Berre@sintef.no, Dumitru.Roman@sintef.no,
hansaage@gmail.com}

Abstract. An important challenge in today's software development domain is the migration of monolithic legacy applications to be provided as Software as a Service (SaaS) in the Service cloud, potentially with execution support through a Platform as a Service (PaaS) cloud computing platform. We present a generic methodology which shows how to migrate legacy applications to the service cloud computing platform. A case study for scientific software from the oil spill risk analysis domain is described, with highlights of current challenges.

Keywords: Legacy systems, Web service, Cloud Computing, MAD, ADM

1 Introduction

Legacy systems are software systems that have been built for more than decades with old technologies and methodologies. These systems are continued to be used today and play critical roles in their domain. When new requirements come to these legacy systems, people find they are hard to maintain and update [1].

As a principle we can talk about 3 layers in a service cloud architecture, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The legacy system we focus on in this paper is in the SaaS level, which features a complete application offered as a service on demand. A single instance of the legacy software runs on the cloud and services multiple end users or client organizations. These SaaS level service clouds runs above the PaaS clouds like Amazon EC2. Compared to traditional way of deploying a system, cloud computing has advantages like incremental scalability, agility, high reliability, high fault-tolerance, service-oriented, and etc [2]. For the above reasons, many IT departments consider to migrate legacy application into the cloud computing, hope can keep the functionalities of legacy system without too much modifications of legacy code, and also take the advantages of cloud computing. Most of these migrations focus on specific technologies like Java or .Net [3], so a generic methodology to guide how to migrate legacy system to cloud platform is necessary and important in scientific software development domain.

This paper presents a generic methodology, which helps developers migrating a legacy system to a SaaS cloud computing platform. This seven-step methodology will guide developers through the migration details step by step and improve the

development and delivery of higher quality scientific software with higher productivity and effectiveness.

The rest of the paper is organized as follows: Section 2 gives an overview of the methodology, and briefly introduces each step of the methodology. In Section 3, detailed descriptions of each step are presented. We will discuss how this methodology works in sequential steps like architectural representation, architecture redesign, model transformation, Web service generation, cloud platform selection, and Web service deployment. Then Section 4 will take the OSCAR example system in the SINTEF SiSaS project as an example to apply this methodology and demonstrate how it works. Section 5 concludes the paper.

2 Methodology Overview

We design a generic methodology for the migration of legacy system to Cloud Platform, which is depicted in Figure 1.

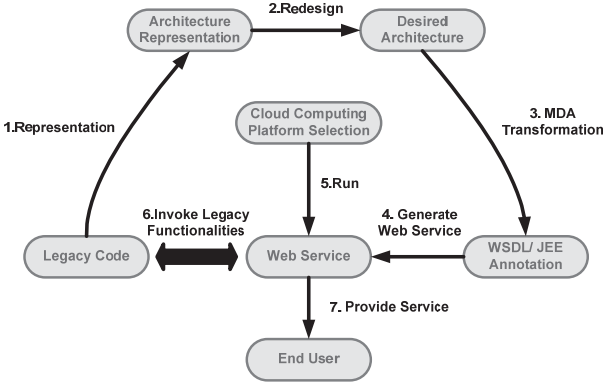

Fig. 1. Methodology Overview.

This generic methodology follows the following steps:

1) *Architectural representation of the legacy application:* Based on the source code and text descriptions, we can analyze the legacy system and reconstruct an architectural model of the legacy application.
2) *Redesign the architecture model:* redesign the original architecture model and in particular identify services that can be provided in a SaaS architecture, specified in a SoaML model [4];
3) *MDA transformation:* with MDA transformation technology [5], we can easily transform the architecture model like SoaML , SysML [6], UML [7] to target code like WSDL, JEE Annotation;
4) *Web service generation:* We can generate the target Web service based on the WSDL or JEE Annotation;

5) *Web service-based invocation of legacy functionalities:* The service-base application invokes the functionalities from identified function and service points in the legacy application;
6) *Selection of suitable Cloud Computing Platform:* According to the specific requirements of target system, the most suitable cloud computing platform will be chosen to support the execution of the Web services;
7) *Web service deployment in the service cloud:* End users can consume the legacy functionalities through the Web services that run on the cloud.

The ideal result is a new Web service application that looks, behaves and maintains workflows just like the old legacy system but will be provided as a SaaS in the cloud.

3 Methodology Guidance

3.1 Representation and Redesign with ADM

Most of legacy systems lack original design documents because of historical reasons, and the only valuable content is the legacy code itself. How to represent the legacy code is a key issue in our methodology. According to the "SEI horseshoe model" shown in Figure 2, the ADM (OMG , Architecture Driven Modernization) reconstruction follows a bottom-up design view and starts from source text to code structure, to function-level, and then to architecture representation.

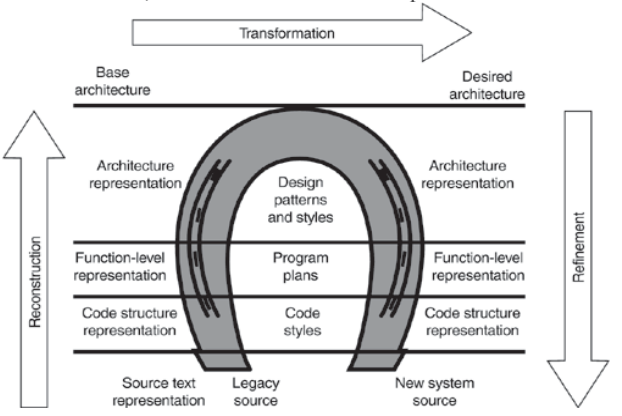

Fig. 2. SEI Horseshoe Model [8].

The legacy applications are usually written with old programming languages like FORTRAN, LISP, and C. These languages are not object-oriented and hard to maintain and re-structure from code level directly. Also we consider how to extract some valuable information, such as business entities and metadata, from legacy system. Figure 3 shows the steps involved in the refactoring.

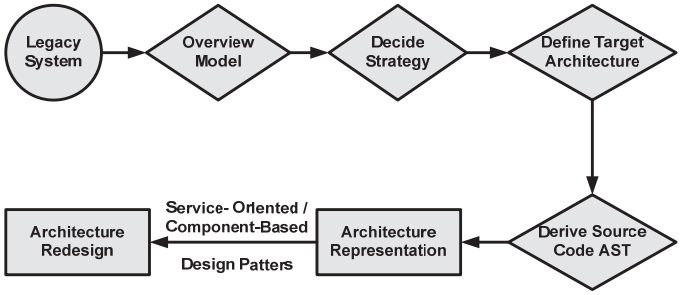

Fig. 3. Refactoring with ADM.

The ADM approach has the following steps: obtain the overview model, select strategy, define target architecture, derive an AST (abstract syntax three) [9] of the source code, create models and profiles (e.g. SoaML, SysML, UML).

For the purpose of running a service in the cloud, we have chosen SoaML as a central modeling language for exemplification. SoaML's target is to help architects and developers to better describe the services at the heart of SOA. The SoaML supports the range of modeling requirements for service-oriented architectures, including systems of services, individual service interfaces, and service implementations. This is done in such a way as to support the automatic generation of derived artifacts following an MDA based approach. With SoaML, we can connect business and technology easily. Beside SoaML, other UML architectural models like SysML are also considered in our methodology, according to specific requirements.

After the representation of legacy system with architecture diagrams, it is necessary to redesign the architecture and rearrange legacy code to meet a service oriented approach. One primary redesign principle is to make the target architecture more service oriented or component-based, which would bring features like high cohesion and low coupling to the target architecture and make it more structural. Different software design patterns, like the GRASP (General Responsibility Assignment Patterns) can be considered in this step.

3.2 Model Transformation and Web Service Generation

In these two steps, we would like to use the redesigned model to create a Web service. The redesigned models like SoaML will be transformed to Web services through MOFScript [10] or transformation tools like ModelPro [11].

ModelPro is an open source MDA provisioning engine form ModelDriven.org community. It is able to read models, and "provision" source code, documents web pages automatically. The current Version of ModelPro is integrated with MagicDraw UML with the Cameo SOA+ Plug-in. SoaML is the first plug-in provisioning cartridge for ModelPro which automates the design and development of Service Oriented Architectures, initially on Java Platforms. With SoaML and ModelPro, enterprise class service oriented architectures can be up and running quickly and

efficiently, ready to integrate new and existing capabilities into coherent solutions based on the enterprise requirements, services and processes.

3.3 Invoke Legacy Functionalities from Web service

A conceptual architecture of how legacy applications is provided as Web service is depicted in Figure 4.

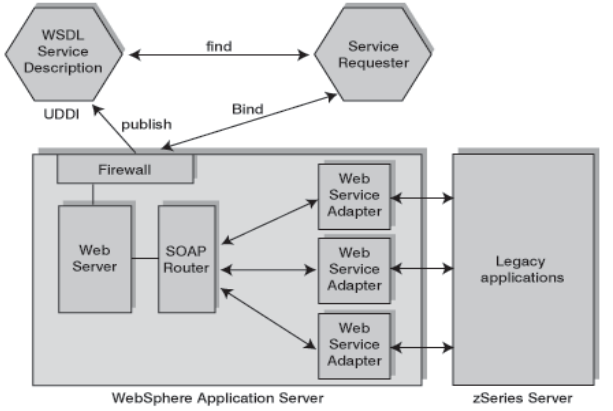

Fig. 4. Legacy Applications to Web service [12].

We can see three main components of Web services in this architecture: service provider, service request, and service broker. When service user requests a service, the WSDL description will be sent to find out the service. Once the service is found, the WSDL description will generate SOAP request message and send to service provider. The service to be deployed on the Web server exposes a number of methods callable from the SOAP client. The service residing on the Web server acts as a client to a legacy application that resides on a different server. The legacy client code that resides on the Web service has a wrapper layer over it. This will facilitate a clean separation from the legacy calls. A Web service adapter (wrapper) is built over the legacy application to expose it as Web service. The legacy client code can be isolated with this kind of design. The wrapper invokes the methods from the legacy application through SOAP.

3.4 Migration to Cloud

In this step we consider to choose a most suitable cloud computing platform to run the generated Web services. The legacy system will run on the cloud at the “Platform as a Service” level. The most popular PaaS level Cloud Computing Platforms today include: Sun Open Cloud Platform [13], Microsoft Azure [14], Amazon EC2 [15], and Google App Engine [16]. The EU project RESERVOIR [17] is also a good choice for providing service on cloud.

To run an application on the cloud is more or less like renting a virtual server and managing code in other organization's platform. We can take other advantages of cloud technologies to make our application more scalable and effective to use. The generic processes of configuring and running a Web service on different clouds are quite similar. Take Sun Open Cloud Platform as an example, the implementation steps include [13]:

1) Choose the running environments from a pre-defined virtual machine images, like load balancer, web server, database server appliances;
2) Configure the running environments to make a custom image, like configuring the load balancer, uploading static content to storage cloud, loading web content from Web Server, deploying database server to generate dynamic content;
3) Program with the redesigned architecture to satisfy specific application requirements;
4) Choose a pattern that takes the images for each layer and deploys them. Also we handle networking, security, and scalability issues.

With this kind of migration steps, it is possible to move the legacy systems to cloud. Moving the Web service to cloud platform doesn't take much modification of legacy code generally.

Different popular cloud platforms are compared in [18]. We could see that it is easier to move a legacy system to Amazon EC2 than to Google App Engine (GAE). Amazon EC2 nearly supports most of the environment, applications, programming languages, runtimes, database servers etc. It offers much more freedom to developers. While with GAE, developers have to use either Python or a JVM-language (like Java), and also have to use the database system provided by Google itself, and communicate with Google database query language. Some tools like Cloud Foundry are used to help us migrating legacy system to Cloud more easily and effectively.

4 Case Study

We applied the methodology we presented in section 2 to the OSCAR system in oil spill risk analysis area. OSCAR (Oil Spill Contingency and Response) is specifically designed to support oil spill contingency and response decision-making. We migrated and rebuilt this legacy application from Client/Server to Browser/Server architecture, and provided this legacy application as a Web service.

Figure 5 shows a screenshot of the OSCAR legacy application with its detail processes description. The "Oil spill processes" contains four main surface processes: oil drift, oil weathering, biological effects, and response options. The OSCAR legacy application simulates how these processes work and how they affect each other.

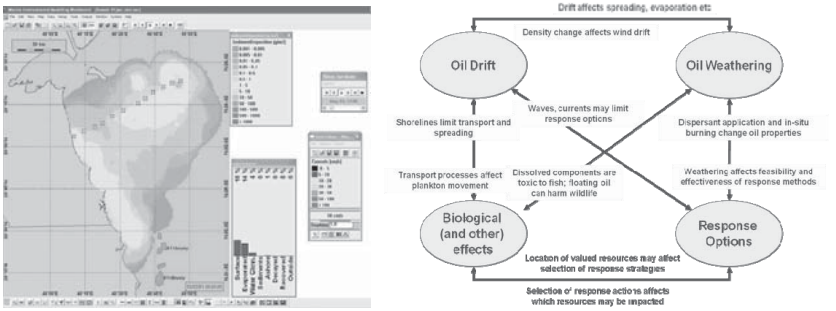

Fig. 5. OSCAR Screenshot.

In step 1, we analyzed the OSCAR legacy code structure, and represented the OSCAR system with a three-layer architecture diagram. The most important business logic is all in the logic layer FATES calculation. FATES engine gives the calculation and simulation supports to the processes like "oil drift" or "oil weathering". The output of step 1 is the architectural representation of OSCAR application, which we can see from the Figure 6.

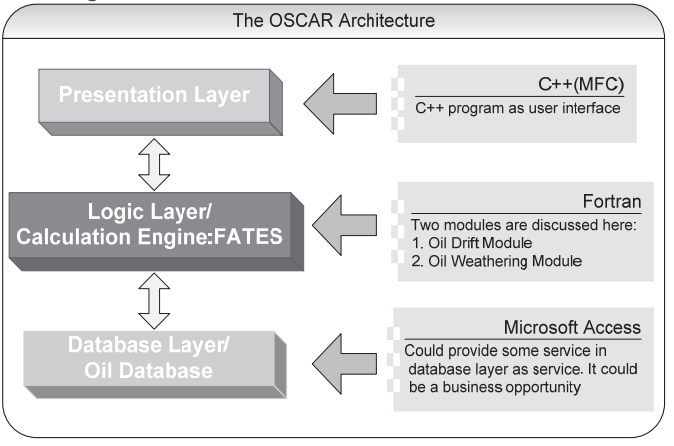

Fig. 6. Overview of OSCAR System Architecture.

Then we redesigned the architecture model in step 2. The target architecture should be service-oriented. The ideal model can separate different modules and provide different modules as a service to each other. We modeled the OSCAR system with SoaML diagrams like Service Architecture and Participant Architecture (See Figure 7). A Services Architecture is a network of participant roles providing and consuming services to fulfill a purpose. The output of step 2 is the redesigned architecture model.

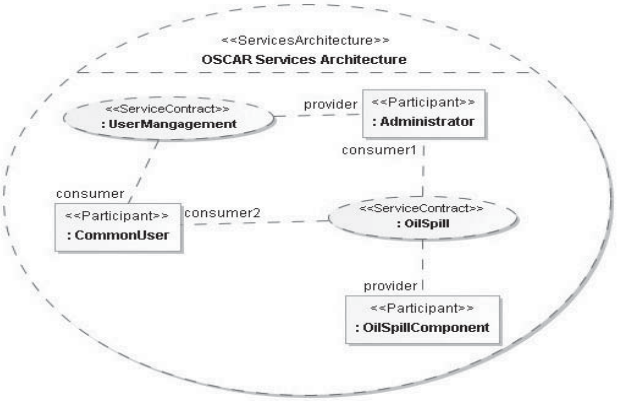

Fig. 7. OSCAR Service Architecture.

In step 3, we generated WSDL with the MDA transformation technology MOFScript according to the redesigned SoaML models. And in coming step 4 we used these WSDL to generate target Web service. The outputs of these two steps are the WSDL, which are addressed in Figure 8.

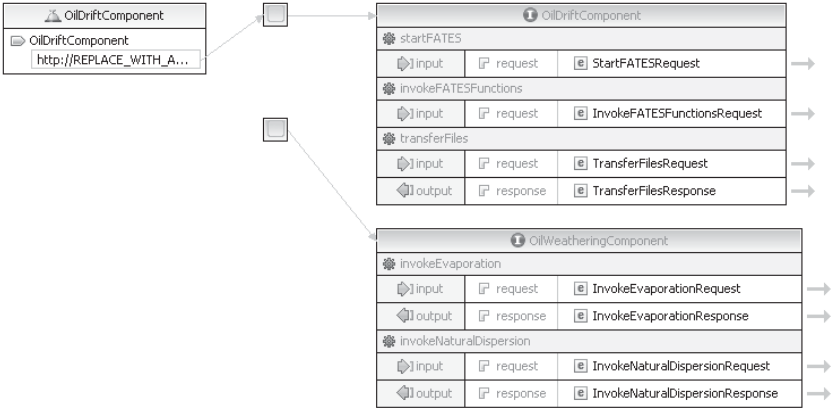

Fig. 8. Model Generated WSDL.

The service-based OSCAR application invokes the functionalities from legacy system modules like oil drift and oil weathering in step 5, which gives the OSCAR Web service application the abilities to provide the functionalities of the legacy application.

We are currently experimenting with different cloud platforms for the execution of this, with an initial trial using Amazon EC2. We intend to compare with other platforms in the future, and also evaluate how the MDA approach can support the adaptation that might be required for different cloud platforms.

5 Conclusion

This paper discussed a generic methodology for migrating legacy systems to the service cloud. It included the following steps: representation of the legacy application, redesign the architecture model with identified services, MDA transformation, Web service generation, invocation of legacy functionalities, selection of a suitable cloud computing platform, and provision of cloud Web service to the end users. We used the OSCAR legacy system as an example to explain how to apply this methodology. Challenges we see in this approach is how to best provide automated tool support for the migration methodology for the different types of legacy systems and programming languages, and for the identification of service-oriented refactoring and migration points. There are also separate challenges associated with the market mechanisms and discovery of SaaS offerings, as well as for the selection and use of appropriate cloud platforms.

References

1. David Woollard, Chris Mattmann, Nenad Medvidovic: Injecting software architectural constraints into legacy scientific applications, pp. 65--71, SECSE (2009)
2. Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic. :Cloud Computing and Emerging IT platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Vol. 25, pp. 599--616, Elsevier Science (2009)
3. Yu Ping, Jianguo Lu, Terence C. Lau, Kostas Kontogiannis, Tack Tong, Bo Yi: Migration of legacy web applications to enterprise Java environments net. data to JSP transformation, pp. 223--237, IBM Press (2003)
4. Object Management Group (OMG). Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS) (2008)
5. M. B. Kuznetso, UML model transformation and its application to MDA technology, Vol 33, pp. 44--53, Programming and Computing Software (2007)
6. Edward Huang, Randeep Ramamurthy, Leon F. McGinnis, :System and simulation modeling using SysML, pp. 796--803, Winter Simulation Conference (2007)
7. Dan Pilone, Neil Pitman. UML 2.0 in a Nutshell, pp.15--27, O'Reilly Media, Inc, (2005)
8. Robert C. Seacord, Daniel Plakosh, Grace A. Lewis, :Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices, 1st edition, pp.58, (2003)
9. Klaus Meffert, Ilka Philippow,: Supporting Program Comprehension for Refactoring Operations with Annotations, Frontiers in Artificial Intelligence and Applications, Vol. 147, pp. 48--67 (2006)
10. A. Z. Javed, P. A. Strooper, G. N. Watson, :Automated Generation of Test Cases Using Model-Driven Architecture, International Conference on Software Engineering (2007)
11. ModelDriven.org. SoaML/ModelPro Tutorials. SoaML Cartridge. http://portal.modeldriven.org/content/soamlmodelpro-tutorials (2009)
12. Dietmar Kuebler, Wolfgang Eibach, :Adapting legacy applications as Web services, IBM, (2002)
13. Sun Open Cloud Computing Document, Introduction to Cloud Computing architecture, Sun Microsystems, Inc.(2009)
14. Azure Service Platform, Micrsoft Corportation, http://www.microsoft.com/azure/services.mspx