Moving Integrated Product Development to Service Clouds in the Global Economy
J. Cha et al. (Eds.)

doi:10.3233/978-1-61499-440-4-307

Service-Oriented Architecture for Cloud Application Development

Hind BENFENATKI[a], Gavin KEMP[a,1], Catarina FERREIRA DA SILVA, Aïcha-Nabila BENHARKAT[a] and Parisa GHODOUS[a]
[a] *University of Lyon 1, INSA - Lyon, LIRIS, CNRS, UMR5205, F-69621, France*

Abstract. Software engineering used several approaches for the development of application such as service oriented approaches. Nowadays, with the advent of cloud computing and the convergence toward "Everything as a Service", application development is moving to a new paradigm, abstracting the underlying architecture and infrastructure. The literature does provide some work describing frameworks and architectures for cloud software development, but not one that covers the whole application development lifecycle. Furthermore, these papers are mainly dedicated to developers and do not provide a business stakeholder a method or an easy to use service to deploy their business application without the help of an IT-professional. Our work fits into the perspective of defining a Service-Oriented Architecture for Cloud Application Development. The architecture we propose is designed for non-IT professional users. It avoids the huge technical background needed for cloud application development by automating the process of development; avoids PaaS dependency and advocates the implicit collaboration by reusing and composing services. This article will give a proposed architecture for this objective as well as an example of its implementation.

Keywords. Cloud Computing; Business applications development; Requirement expression; Linked services; Services reuse

Introduction

The increasing complexity of software systems and the constant expansion of new requirements require the cooperation of many professional skills. With the web 2.0 and cloud computing, collaboration through the use of third party services has emerged; indeed, the composition of services allows implicit collaboration between different software entities and thus between different partners.

Web and cloud services are a popular medium for application development and deployment on the cloud. Modern enterprises are moving towards cloud service-oriented architectures to promote reuse and interoperability of services; and to benefit from the cloud computing advantages, such as small initial investment, no license acquisition, accessibility from everywhere and every time, high availability and so on.

Cloud applications are nowadays developed in Platforms as a Service (PaaS) and deployed on virtual infrastructures. Cloud applications are referred as Software as a Service (SaaS) which are service oriented, and distributed. Application development is different from one PaaS to another. In fact, each PaaS offers several Application

[1] gavin.kemp@etu.univ-lyon1.fr

Programming Interfaces (APIs) and has its own architecture for storing data and deploying instances. The underlying infrastructure is abstracted from the user.

Several research work describing cloud application development are mainly dedicated to developers, and do not allow a non IT-professional to develop application. In this paper, we describe architecture for business application development for cloud environments allowing business stakeholders to proceed to automatic development which promotes service reuse. The service discovery and composition processes are done from user's request that describes functional and non-functional business application requirements. Functional requirements describe service features. Non-functional requirements describe user preferences and Quality of Service (QoS) parameters. We describe the implementation of our approach.

The rest of this paper is organized as follows. Section 1 describes the work related to existing cloud software development approach. Section 2 presents the architecture of our approach and section 3, its implementation. Section 4 draws final conclusions and describes our future work.

1. Related Work

In cloud computing paradigm, there is a lack of complete application development methodologies. However, several partial approaches for the development of applications exist in literature. In [1], the authors propose an approach that uses Domain Specific Languages (DSL) within the process of development and deployment of software on the cloud. The main inconvenient with this approach is the huge time that consumes the DSL development in early phase of their approach.

In [2], the authors describe a methodology for cloud-native application design, which considers CAP (Consistency, Availability and network Partitioning tolerance) parameters, and present a framework instantiating this methodology. The main lack of this methodology is that it focuses only on the CAP properties to the detriment of QoS parameters (such as response time and security indicators) for designing an application and choosing cloud services and does not describe how the development and deployment of the application are done.

Giove and colleagues [3] propose a library called CPIM (Cloud Provider Independent Model) offering PaaS level services such as message queues, noSQL services, and caching service, abstracting from the details that are specific of the underlying PaaS provider; and allowing an application developer to implement his application in a PaaS independent way. At deployment time, the developer specifies the PaaS to be used. At runtime, CPIM library acts as a mediator between the application code and the services offered by the PaaS. Our work will reuse and integrate this interesting approach for developing undiscovered services.

In [4], the authors propose the MODACLOUDS system, a European project [5] that uses the principle of MDD (Model Driven Development) for the development of applications on the cloud. Applications are designed at a high level of abstraction of the target cloud, making them capable of operating on multiple cloud platforms. The main lack in this work is that the cloud services selection is not taking into consideration the platforms APIs and services in the process of choosing the best PaaS cloud providers, but only their QoS parameters.

The work proposed by [6] describes a SaaS Development Life Cycle (SaaSDLC). The authors present an approach that promotes evaluation of the cloud provider based on capabilities of a platform. The SaaSDLC does not consider reuse of cloud services. It promotes the development to a specific platform, making application portability more difficult.

In [7], the authors advocate the intervention of cloud provider in the Agile eXtreme Programming software development process, especially in planning, designing, building, testing and deployment phases to mitigate the challenges associated with cloud software development, and make it more advantageous. In this paper, the authors integrate the notion of roles for the various stakeholders in the agile development process for cloud applications, but do not consider the other characteristics of cloud applications that can influence the development process.

In [8], the authors describe Service-Oriented Software Development Cloud (SOSDC), a cloud platform for developing service-oriented software and a dynamic hosting environment. The SOSDC adopts an architecture covering the three levels of cloud services. The IaaS level is primarily responsible for providing infrastructure resources. The PaaS level provides App Engine for testing, implementing and monitoring the deployed application without having to consider the technical details. SaaS level aims to provide "Online Service-Oriented Software Development Environment" and includes the two following modules: Xchange – a service supporting shared web services – and MyCloud, a personal development environment for each developer. Once an application is built, the developer may request an App Engine hosting environment by specifying the deployment requirements. This approach aims to supply a dynamic development environment by providing on demand appliance for developers, but it is dedicated to a specific platform and does not exploit public cloud platforms.

In summary, the state of the art analysis shows that most approaches in the area of cloud application development are dedicated to developers. The approach we propose in the next section, (i) obeys the SOA principles and technics that promote the reusability, the loose coupling and the composability of the underlying Everything as a Service (XaaS), (ii) maintains interoperability through the use of cloud services and the modelling of functionalities to be developed; (iii) meets the requirements of the distributed nature of cloud; (iv) aims to make software development more accessible for non IT-professionals; and (v) is independent of a specific platform.

2. The Proposed Architecture

This section describes our architecture for cloud application development (Figure 1). This architecture follows the MADONA's methodology (Methodology for semi-Automatic Development of clOud-based busiNess Applications) [9] which is based on SOA principle and covers the whole application development lifecycle, from the requirements expression to the tests and validation phases.

It combines service discovery and composition, with service development using cloud platforms, when the discovery process does not return a service meeting the user's requirements. We use a cloud service orchestration tool which allows an easy deployment, and dependencies management of the deployed services. The approach we propose reduces cloud provider dependency, by reusing cloud business services, and

developing undiscovered services with MDD abstracting cloud platforms constraints. The primary goal of our approach is to allow a business stakeholder to automatically develop a cloud-business application simply by describing his/her business requirements via a web form.

2.1. Project Management

The business stakeholder enters his requirement in a web form to generate a file, based on Linked-USDL [10], [11], [12], describing functional and non-functional information on the needed cloud business application we called .rival.

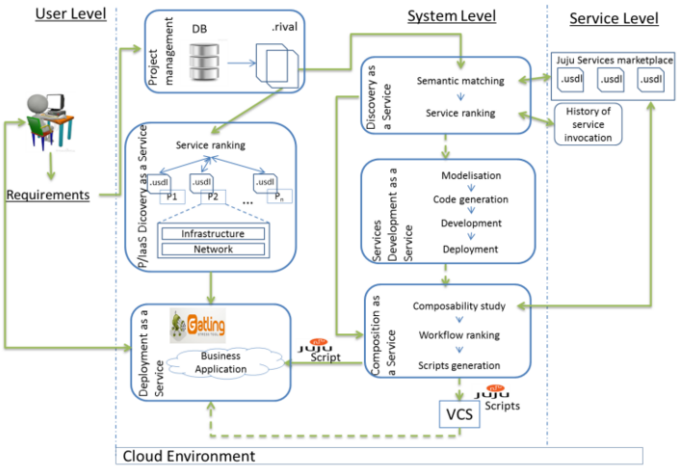

Figure 1. Architecture for cloud application development

2.2. Discovery as a Service

The service discovery consists of matching the user's requirements with the cloud marketplace's services. The user's requirements are expressed via .rival files, and marketplace's services are described using .usdl files based on Linked USDL principles. The marketplace's services description include the following information : (i) the service name, (ii) the service description, (iii) the service classification (SaaS, PaaS, IaaS), (iv) the hard composition constraints, i.e. the specific services that must be composed with the one described, e.g. WordPress has MySQL as an imposed database to function, (iv) the soft composition constraints, i.e. the family of services that have to be composed with the one described, e.g. SugarCRM has to be composed with a database but has no imposed database, thus it can be composed with, for example, MySQL or Oracle, (vi) the composition possibility, i.e. the services that can be composed with the one described, e.g. a CRM can be composed with a mailing service.

2.2.1. SaaS discovery:

SaaS discovery consists of discovering a SaaS from the marketplace meeting the user's requirements. Like illustrated in the Figure 2, the SaaS discovery follows these steps: first, we check if the stakeholder has a preferred provider for a given service, if so, we

select the desired function supplied by this same provider, else, we check respectively the matching between requirements and services according to (i) user preferences, namely, service location and purchase details, (ii) supplied functions and composition constraints, and (iii) QoS requirements. If the deployment of matched services requires the deployment of another service, the SaaS discovery process restarts for the service that has to be composed with the one matched the user's desired function; e.g. in the context of the CRM SaaS, when matching several CRM services, we note (from the .usdl file) that we have to compose a database with the CRM service. The database service discovery is therefore performed. Our SaaS marketplace is represented by the services of a cloud services orchestration tool, because it simplifies the dependencies management, and the deployment of services. For our preliminary tests, we use "Juju" [13], a cloud services orchestration tool, which allows the use of charms for the deployment and dependencies management of supplied services. We describe the "Juju" marketplace services using .usdl files which will be used for matching user's requirements.

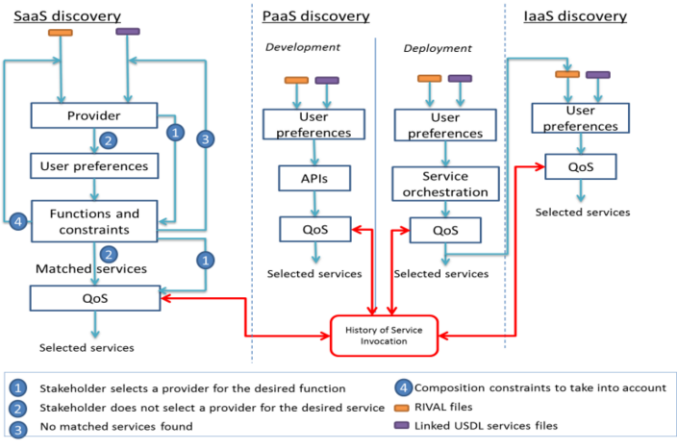

Figure 2. Service Discovery process

2.2.2. PaaS discovery:

PaaS discovery consists of discovering a PaaS for the deployment of a resulted cloud business application and/or for the development and the deployment of undiscovered services i.e. when no matched service is found for a desired function. The PaaS discovery (Figure 2) for the development and the deployment of undiscovered services is done by matching the user requirements (.rival files) with several PaaS description (.usdl files) according to user preferences; APIs offered by PaaS and QoS requirements. The PaaS discovery for the deployment of the resulted cloud business application is done by matching the user requirements (.rival files) with several PaaS description (.usdl files) according to the user preferences, the service orchestration tool supported by the PaaS (allowing an easy service composition), and the QoS requirements.

2.2.3. IaaS discovery:

A cloud infrastructure is selected if the PaaS discovery process for the deployment of the business application does not return a matched platform supplying the needed

services. The automatic IaaS selection is performed according to user's preferences and QoS requirements.

We consider matched marketplace's services for a given desired service as equivalent. We rank matched services according to QoS indicators (Response time, Availability, Accessibility, Security, etc.) and assigned QoS coefficients. In fact, the user should assign coefficients to QoS parameters based on its own priorities. We consider a "history of service invocation" that provides us service QoS parameters according to previous service invocation. The service ranking and selection is described in the next section.

2.3. Service selection :

The service ranking is calculated based on the coefficients associated to QoS attributes and assigned by the user based on its priorities such as the sum of all the coefficients equals 10. The service with the highest rank will be selected. Two scenarios are available: Let S_i be a service and Q_i a QoS indicator.

$$R(S_i, Q_i) = \begin{cases} R_{upper} \\ R_{lower} \end{cases} \quad (1)$$

Case 1: the higher the value of the attribute is, better is the service, for instance, the service availability. In this case the rank associated with this attribute for a given provider is calculated as follows:

$$R_{upper} = \frac{Value}{Max} * Coefficient \quad (2)$$

Where: Value is the value of the attribute for a given provider. Max is the maximum value of the attribute among all providers. Coefficient is the coefficient previously assigned to the attribute by the stakeholder.

Case 2: the smaller the value of the attribute is better is the service, for instance, the response time. In this case the rank associated with this attribute for a given provider is calculated as follows:

$$R_{lower} = \left(1 - \frac{Value}{Max}\right) * Coefficient \quad (3)$$

Let $R(S_i)$ be the global ranking regarding the whole indicators for a service S_i.

$$R(S_i) = \sum_{j=1}^{n} R(S_i, Q_j) \quad (4)$$

For equivalent services, the service with the highest rank is selected. The service discovery continues even after deployment to allow generating new compositions that can be better that the one deployed.

2.4. Service Development as a Service

Development of undiscovered services, i.e. no service has been discovered for a desired function, are developed using a MDD approach following four key steps:

- Modelling: Undiscovered services are modelled with UML notation abstracting the deployment PaaS making the modelling reusable and independent from PaaS.
- Code generation: A PaaS platform is selected for business service deployment according to the PaaS selection method described in section (2.2.2). PaaS dedicated code is generated from UML diagrams.
- Coding: The generated classes have to be completed in order to achieve the desired functionality. This has to be done with the intervention of a developer.

- Deployment: At execution time the developed service is deployed on a preselected PaaS, so that it can be invoked.

2.5. Composition as a Service

We compose selected and developed cloud services. We use a cloud service orchestration tool which allows an easy deployment and dependencies management of services. Before the composition is done, we analyse the composability of selected services starting by those having the highest rank. The composability study takes into account the services that have to be composed, and the ones that can be composed with the given service. The composition constraints and possibilities are described for every marketplace's service throw the .usdl file. With our approach, these constraints are depicted in the service description (.usdl file) and not in the user requirement in order to automate the services dependencies management, and avoid the user to detail his requirements. The composability study helps us to generate the composition workflows and their corresponding scripts by considering affinities and constraints between services, and global criteria like maximum cost of the application deployment. The role of the generated script is explained in the section (2.6). Several versions of workflow composition are stored in VCS (Version Control System). The composition with the highest rank is deployed on a preselected PaaS or IaaS. We reserve the other versions in the event that the stakeholder does not validate the deployed business application after tests. For the selected workflow, several web interfaces allowing the configuration of the generated application are displayed to the user so that he can personalize the application by integrating information related to its business such as choosing a logo or a name for his service.

2.6. Automatic deployment

The deployment process concerns the deployment of the resulted business application composing discovered and developed services. Two types of deployment are considered:

1) On preselected PaaS: The deployment is done by injecting a script deploying the composed services involved in the selected workflow.

2) On preselected IaaS: The deployment is done by injecting a script installing the orchestration services environment and deploying the composed services involved in the selected workflow.

For both cases, the script corresponds to a dedicated cloud services orchestration tool ("Juju" in our case) command lines allowing services manipulation (deployment, dependencies management). "Juju" environments can be bootstrapped on many clouds: Amazon [14], OpenStack [15], and so on. A script specific to the platform can be generated. Redeployment can occur after the tests and validation phase, if the stakeholder does not validate the resulted business application after the tests (performance and conformity) have occurred. In this case, allocated resources for the previous deployment of the business application are freed and another composition is deployed.

2.7. Tests and Validation of the deployed business application

The validation is done by the business stakeholder after testing the deployed business application. Two types of test are considered: performance tests and conformity tests. Performance tests are done automatically using Gatling tool [16], an efficient open source load testing tool. Conformity tests are done by business stakeholder, where he tests the correspondence between his/her requirements and the resulted business application. After tests, the stakeholder notifies to the system his/her positive or negative validation result. If the validation result is negative, another composition from the VCS is deployed, the tests are performed, and the stakeholder has to notify his/her validation results. This cycle is repeated until the stakeholder satisfaction is achieved or no other composition is possible.

3. Implementation of our architecture

The implementation of our approach is done using Grails framework and a MVC (Model View Controller) architecture, respectively coded in java, gsp and groovy.

3.1. Project management:

The Business stakeholder enters, on a web form, the description of the needed service; the requirement of location, currency, price, payment, provider and QoS. The .rival file is generated using the jena API. This file is then stored for future use or is immediately read for the service discovery. Discovery as a Service:
The service discovery consists of extracting the requirement information from the .rival file using SPARQL query, service provided by the jena-arq API; then adding the outputs to a new SPARQL query and applying it to all the known .usdl service description files; then selecting only those that return a value to our request. This second SPARQL request returns also the hard constraints and soft constraints needed for the service composition.

3.2. Composition as a Service:

Services, often, do not work alone and have to use other services to work. Thus WordPress must work with a MySQL database; this is defined as a hard constraint because this is an imposition from the WordPress service; on the latter SugarCRM must have a database; this is defined as a soft constraint because you have a choice for your database e.g. MySQL or Oracle.

If the stakeholder needs a blog engine (Figure 3), the .rival file is generated and is matched to all known .usdl files. This returns WordPress and BlogEngine.NET both use hard constraints, thus the information extracted from the constraints goes throw a strict comparison with usdl:name of the .usdl files to return MySQL for WordPress and Oracle for BlogEngine.NET. This process is repeated as long that there are constraints needed.

If the stakeholder needs a CRM service (Figure 4), the .rival file is generated and is matched to all known .usdl files. This returns SugarCRM and VTigerCRM which both use soft constraints, thus the information extracted from the constraints goes throw a

flexible comparison with usdl:hasDescription of the .usdl files and thus both return MySQL and Oracle. This process is also repeated as long that there are constraints needed.

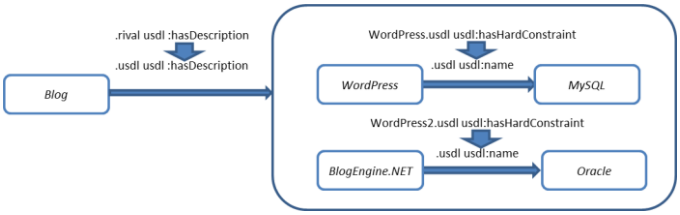

Figure 3. Composition tree for a blog engine

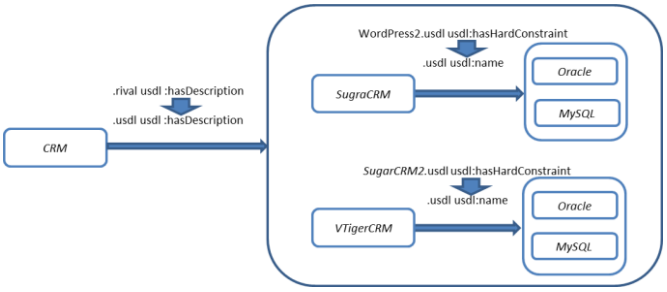

Figure 4. Composition Tree for a CRM service

From the Service composition, several workflows can be generated. These workflows need to be ranked to select the best depending on the user QoS requirements.

3.3. Service Ranking:

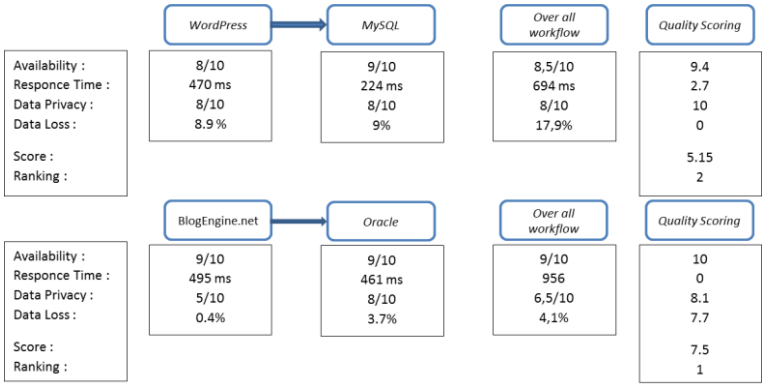

Figure 5. Ranking the workflows for a blogging engine

Once the workflows are generated, the QoS files of the individual services in a workflow are read then added to the total scoring of each workflow. Each QoS parameter is scored according to equations defined in section 2.3.

In the case of the blogging engine, the QoS requirements are defined as:

Availability= 2, Response Time= 1, data loss= 3, data privacy = 4.

Availability and data privacy will use equation (2) since the higher the score the better and response time and data loss will use equation (3).

From Figure 5, we observe a better quality for the second workflow using BlogEngine.NET.

The workflows are sorted according to their rank and stored in case of negative validation results. A script deploying and connecting all needed services with juju is generated for the best workflow.

4. Conclusion and Future Work

In this paper we presented the architecture and implementation of our approach for cloud application development, an agile approach of service discovery and deployment designed for those with little background knowledge on cloud based services. This means that, the people closest to the business project can deploy the cloud services they need with little intervention of an exterior IT professional. Our approach uses services, described using Linked-USDL, available for service orchestrator such as Juju for automatic deployment.

Future developments include mainly the implementation of a charm [15] creation tool and the performance tests scenarios generation according to the Gatling tool template.

References

[1] K. Sledziewski., B. Bordbar and R. Anane, A DSL-based Approach to Software Development and Deployment, 24th IEEE International Conference on Advanced Information Networking and Applications, 2010
[2] V. Andrikopoulos., C. Fehling and F. Leymann, DESIGNING FOR CAP: The Effect of Design Decisions on the CAP Properties of Cloud-native Applic0ations, CLOSER 2012. 2nd International Conference on Cloud Computing and Services Science. Proceedings, 2012
[3] F. Giove., D. Longoni., M. Shokrolahi Yancheshmeh., D. Ardagna and E. Di Nitto, An Approach for the Development of Portable Applications on PaaS Clouds., CLOSER 2013 - 3rd International Conference on Cloud Computing and Services Science, 2013
[4] D. Ardagna., E. Di Nitto., G. Casale., D. Petcu., P. Mohagheghi., S. Mosser., P. Matthews., A. Gericke., C. Ballagny., F. D'Andria., C.-S. Nechifor and C. Sheridan, «MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds,» MiSE 2012, 2012
[5] MODACLOUDS, Available : www.modaClouds.eu/
[6] H. Kommalapati and W. H. Zack, «The SaaS Development Lifecycle,» Available: www.infoq.com/articles/SaaS-Lifecycle., 2011
[7] R. Guha and D. Al-Dabass, impact of Web 2.0 and Cloud Computing Platform on Software Engineering,» IEEE, International Symposium on Electronic System Design, 2010
[8] H. Sun., X. Wang., C. Zhou., Z. Huang and X. Liu, Early Experience of Building a Cloud Platform for Service Oriented Software Development, 2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010
[9] H. Benfenatki, C. Ferreira Da Silva, N. Benhartka, and P. Ghodous. Cloud Application Development Methodology, IEEE/WIC/ACM International Conference on Web Intelligence, 2014
[10] Linked USDL, Available: www.linked-usdl.org/
[11] LinkedData, Available: linkeddata.org/
[12] C. Pedrinaci, J. Cardoso, and T. Leidig, Linked USDL: A Vocabulary for Web-scale Service Trading, In 11th Extended Semantic Web Conference (ESWC), 2014
[13] Juju, Available: juju.ubuntu.com/
[14] Amazon EC2, Available: aws.amazon.com/ec2/
[15] OpenStack, Available: www.openstack.org/
[16] Gatling Tool, Available: gatlingtool.org/
[17] Juju chams, Available: https://juju.ubuntu.com/docs/charms.html