Automatic Software Development as a Service (ASDaaS)

Hind Benfenatki[1], Catarina Ferreira Da Silva[1], Nabila Benharkat[2], and Parisa Ghodous[1]
[1]*Université Lyon 1, LIRIS, UMR5205, F-69622, Lyon, France*
[2]*INSA Lyon, LIRIS, UMR5205, F-69621, Lyon, France*

Keywords: Cloud Computing, Business Applications, Requirement Expression, Linked-Data.

Abstract: Cloud-based services have become a norm for business application development. With Cloud Computing and the convergence toward Everything as a Service (XaaS), we no longer consider the classical context of application development, where IT teams or integrators are solicited. Current approaches in Cloud environments are usually designed for a specific Cloud platform; moreover, they are only designed for technical users. To overcome the lack of generic and complete methodology for business application development, we propose a methodology for Automatic Software Development as a Service (ASDaaS), which is designed for non-technical users and promotes services reuse. In this paper, we focus on the phase of business software requirement gathering of our methodology. We define the requirement vocabulary based on linked data principles, and extend the Linked-USDL language to describe business stakeholder requirements as service functions, business constraints, user preferences and QoS parameters. Our approach is illustrated with an e-commerce example.

1 INTRODUCTION

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.

With Cloud Computing and the convergence toward Everything as a Service, we no longer consider the classical context of application development, where IT teams or integrators are solicited to perform software development. We propose a business application development process that minimizes business stakeholder intervention and meets Cloud characteristics.

Let us consider an e-commerce scenario in which a business stakeholder in a European company ”A” would automate the IT hardware acquisition process. The business stakeholder wants to make available to its staff a Cloud hosted business application that allows the following functions: (i) to select products with the edition of purchase orders, (ii) to proceed to the online purchase, (iii) and to make delivery of the product. The business stakeholder has to describe functional and non-functional requirements. Functional requirements include application business functions and business constraints. One of the business constraints is the budget, and for this ”A” imposes a maximum purchase order value equal to 1000 euros. Non-functional requirements include user preferences and QoS Parameters. The business stakeholder prefers to use ”PayPal Service” for the online purchase. For the deployment of the business application, the company ”A” relies on previous good experience using ”Amazon EC2” (http://aws.amazon.com/fr/ec2/) and prefers to deploy its business application on this same platform. Company ”A” prefers to pay deployment costs with the European currency ”euro”. This scenario requires great caution with respect to data privacy and data integrity due to the online payments service. For maintaining the confidentiality of their data, ”A” prefers invoked services be located in European region.

Based on our scenario, we identify the following three challenges, which are addressed in the paper: (i) how to develop a Cloud business application that meets the business stakeholder requirements with minimal intervention and that leverages the Cloud

Benfenatki H., da Silva C., Benharkat N. and Ghodous P..
Automatic Software Development as a Service (ASDaaS) .
DOI: 10.5220/0004844400950102
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 95-102
ISBN: 978-989-758-019-2

benefits. Nowadays, there is a lack of a standard approach for Cloud applications development, as this research aspect is still in progress, (ii) how to specify and model the functional and non-functional business requirements. There is a need for a model that provides explicit description of requirements to ensure then the correct selection of services, and (iii) how to adapt or extend service description languages to facilitate services selection that meet user's requirements.

In this paper, we propose a methodology for business application development for Cloud environments allowing business stakeholders to perform the automatic development of those applications. This methodology is called Automatic Software Development as a Service (ASDaaS) and promotes the discovery and the composition of Cloud services that match functional and non-functional business application requirements. Functional requirements describe service features and business constraints. Non-functional requirements describe user preferences and Quality of Service (QoS) parameters.

The originality of our work presented in this paper lies in the business application requirement description phase of the methodology. We define the requirement vocabulary based on the linked data principles (http://www.linkeddata.org/), and extend the Linked USDL language (http://www.linked-usdl.org/) so that it can describe business stakeholder's requirements as service functions, business constraints, user preferences and QoS parameters.

The rest of this paper is organised as follows. Section 2 describes the works related to existing Cloud software development methodologies. Section 3 presents the proposed ASDaaS methodology. Section 4 introduces ASDaaS's architecture. We describe the implementation and evaluate our work in section 5. Section 6 draws final conclusions and describes our future work.

2 RELATED WORKS

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 (Sledziewski et al., 2010), the authors propose an approach based on Domain Specific Languages (DSL) within the development process. The main inconvenient with this approach is the huge time that consumes the DSL development in early phase of their approach.

Giove and colleagues (Giove et al., 2013) propose a library called CPIM (Cloud Provider Independent Model) 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 (Ardagna et al., 2012), the authors propose the MODACLOUDS system, a European project (http://www.modaClouds.eu/.) 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 PaaS Cloud services selection is only taking into account QoS parameters to the detriment of the platform's Application Programming Interfaces (APIs).

The work proposed by (Kommalapati et al., 2011) 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 (Guha et al., 2010), 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. These 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 (Sun et al., 2010), 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". This approach aims to supply a dynamic development environment by providing on demand appliance for developers, it is dedicated to a specific platform and does not exploit public Cloud platforms.

In summary, the state of the art analysis shows there is a need for a complete methodology that guides Cloud-based developers to perform the development of Cloud service-oriented business applications, considering the constraints and benefits of Cloud environments.

The methodology we propose in the next section, (i) obeys the SOA principles and techniques that promote the reusability, the loose coupling and the composability of the underlying XaaSs; (ii) meets the requirements of the distributed nature of Cloud; (iii) aims to make software development more accessible for non IT-professionals; and (iv) is independent of a particular platform.

3 AUTOMATIC SOFTWARE DEVELOPMENT AS A SERVICE METHODOLOGY (ASDaaS)

Our methodology for Automatic Software Development as a Service (ASDaaS) is aimed at business stakeholders who need a business application to automate a frequent task. In this section, we describe the ASDaaS methodology through the described scenario.

The originality of our approach lies in the following: (i) it allows selection of a development platform (PaaS) according to user preferences and restrictions of available Cloud services, such as suitable APIs, and (ii) it allows selection of IaaS for deployment that meets the predefined user preferences and QoS parameters.

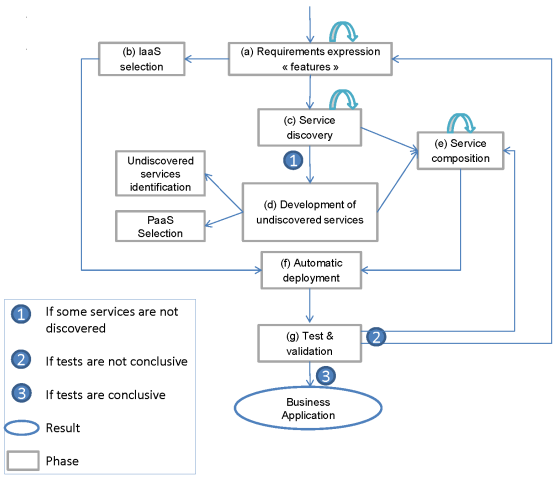

Figure 1: Automatic Software Development as a Service methodology (ASDaaS).

Figure 1 illustrates the different phases of the methodology, which are the following:

a) Requirements expression;

b) IaaS selection for application deployment;

c) Service discovery;

d) Development of undiscovered services;

e) Service composition;

f) Automatic deployment of business application;

g) Tests and Validation of the deployed business application.

These phases are explained in the next sections. In this paper we focus on business application requirement expression, i.e. the initial phase of the ASDaaS methodology.

3.1 Requirements Expression

Services description languages such as WSDL (Web Service Description Language) focus on the description of functional aspects of software component interfaces (Coulouris et al, 2011) to the detriment of non-functional aspects.

The USDL (Unified Service Description Language) was created to address WSDL shortcomings by describing business, operational and technical aspects in services description. Linked USDL consists on reuse of existing vocabularies and ontologies for services description. The rationale behind the use of Linked Data is the requirement that USDL descriptions should be shared between interested parties and linked to other descriptions, standards and formats (Cardoso et al, 2013).

We extend Linked USDL with a new module enabling to describe business application requirements. This way it will be easier to match business stakeholder's requirements with Cloud services described in Linked USDL.

The business stakeholder describes his/her business application requirements (functional and non-functional). Functional requirements are the application features and the business constraints imposed by the stakeholder's organization. Non-functional requirements are the QoS parameters and stakeholder preferences. The latter describes, among others, the location of services an payment means. The QoS parameters describe the stakeholder's business application requirements in terms of quality of service of Cloud platforms and infrastructures.

Following the predefined scenario, the stakeholder defines his/her business application requirements as follows:

- Application functions:
  - Laptop review service,

  - Computer purchase service,
  - Delivery service.
- Business constraints:
  - Purchase cost max value = 1000 Euros.
- QoS parameters: in order to describe its priorities regarding QoS parameters, he/she should assign coefficients to QoS parameters so that the sum of all the coefficients does not exceed 10. As business application integrates online payment, the stakeholder prefers to value data privacy and integrity more than service response time with the following coefficients:
  - Data privacy: 3,
  - Data integrity: 3,
  - Availability: 3,
  - Response time: 1.
- Stakeholder preferences:
  - Services location: Europe,
  - Currency: Euro,
  - Cost max value: 500,
  - Preferred deployment provider: ”Amazon”,
  - Preferred provider for the purchase service: ”Paypal”.

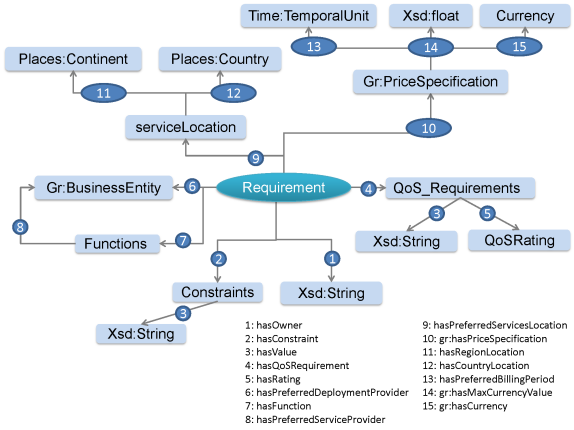

Figure 2: USDL-requirement vocabulary.

The existing USDL modules (https://github.com/linked-usdl.) are not sufficient to describe non-functional aspects of stakeholder business requirements, such as the preference for invoking a business provider rather than another. To achieve the goal of providing a unified language to describe business application requirements, we propose to extend the Linked USDL language by defining a ”USDL-requirement” vocabulary, which will allow matching the business application requirements with the Linked USDL Cloud service descriptions. Multiple vocabularies exist (http://lov.okfn.org/dataset/lov/); we will reuse some of their classes and properties following linked data principles. The USDL-requirement vocabulary describes business application requirements in terms of stakeholder preferences, QoS parameters, service functions, and business constraints. Figure 2 illustrates classes and properties that we define in the USDL-requirement vocabulary. Rectangles represent classes, and the circles represent the properties. For example, the property 7 illustrates that a requirement comprises functions, and the property 8 shows that the stakeholder can define his preferred provider for a given function. The gr:BusinessEntity is a Good Relation’s (Hepp 2011) class that describes, in our context, a service provider.

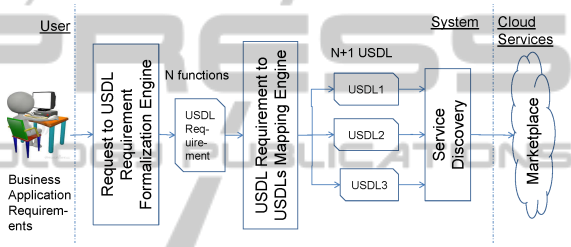

Figure 3: Requirements to USDL translation.

Figure 3 illustrates the matching process from Linked USDL-requirement business application description to Linked USDL Cloud services. Once the stakeholder expresses its business application requirements, these are formalized according to the template of our ”USDL-requirement” vocabulary. This process generates a file that is further divided into several USDL files, each one describing a desirable Cloud service. These USDL files provide input for the Cloud service discovery process. If the USDL-requirement file describes ”N” functions, the USDL-requirement to USDL mapping engine generates ”N+1” USDL files describing abstract services. ”N” USDL files correspond to the ”N” functions, plus one USDL file describing the deployment platform.

Listing 1 illustrates the functions expression using USDL-requirement vocabulary in our scenario. Line 3 presents the hasFunctions property for describing several functions needed by the stakeholder (from line 4 to 6). Lines 8 to 10 show that the stakeholder prefers invoking the paypal service for the computing purchase function. From this listing file, four USDL service files are generated, corresponding to the requirements description of abstract services, which are the products review service, the computer purchase service, the delivery service and the IaaS deployment platform. The service discovery process receives these USDL files as input.

```
1: @prefix rq: <usdl-requirements vocabulary>.
2:
3: rq: hasfunctions
4:  :laptop review service,
5:  :computer purchase service,
6:  :delivery service.
7:
8: :computer_purchase_service a rq:function
9:   rq:hasPreferedProvider
10:  [a gr:BusinessEntity :Paypal]
```

Listing 1: Functions expression with USDL Requirement vocabulary.

3.2 Service Discovery

The process of service discovery starts from the USDL files describing abstract services for outputting a business service or a composite business service.

In this phase, through a business services search engine, we have to discover business services corresponding to several required functions and meeting location, currency and preferred provider criteria. For each service, several services can be discovered and have to be ranked. The advantage of this phase is to reduce the workload and improve reusability of business services. If no business services are discovered for some abstract services, we move on to the phase of service automatic development.

We have to discover services in a Cloud marketplace by matching the USDL describing abstract services and the marketplace's USDL Cloud services. Table 1 illustrates discovered services for the requirements described in our scenario with their QoS indicators and cost. Where Q1 represents data privacy, Q2 is data integrity, Q3 data loss, Q4 access control, Q5 availability, Q6 response time. Values of Q1 to Q5 represent satisfaction percentage of each indicator. We assume that these values are retrieved from a history of invoking these services.

Table 1: Scenario's discovered services.

Needed functions	Discovered services	Q1 (%)	Q2 (%)	Q3 (%)	Q4 (%)	Q5 (%)	Q6 (ms)	Cost (€/ month)
Products review	S1	90	85	0	100	90	10	20
	S2	100	95	2	98	80	20	40
Computer purchase	Paypal	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	32
Delivery	S3	60	50	5	70	60	30	100
	S4	85	90	3	90	50	5	59
IaaS	Amazon	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	26

Currently, for the discovery of existing services, we apply a syntactical matching. The next section describes how the services selection and their composition are done.

3.3 Service Selection and Composition

The service selection phase translates as a multi-criteria decision making problem (Figueira, 2005). Based on stakeholder's preferences and QoS parameters, we weight the discovered services to select a service that meets most requirements and preferences of the user. The service provider with the highest rank will be selected. The rank is calculated based on the coefficients associated to QoS attributes. Two scenarios are available: Let S_i be a service and Q_i an indicator.

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

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. The sum of several coefficients for all attributes does not exceed 10.

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

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

Let $R(S_i)$ be the global ranking regarding the whole indicators for a service S_i. The IaaS provider with the highest rank is selected.

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

In our scenario, the service discovery phase generates a list of services corresponding to different functions described by the user. For each function, the service with the highest rank is selected. The cost is considered as a global criterion. Table 2 illustrates cost and rank calculation for each selected service in our running scenario. Service S1 is selected for the laptop review service requirement. Service S4 is selected for the delivery service requirement. In our scenario, the stakeholder has chosen "Amazon" as an

IaaS provider. The final composition includes the services S1, S4, Paypal and Amazon.

Table 2: Service costs and ranks.

Services	Services rank	Service cost (€/month)
S1	R(S1)=(90/100)*3+(85/95)*3 +(90/90)*3+(1-(10/20))*1 =8.88	20
S2	R(S2)=(100/100)*3 +(95/95)*3+(80/90)*3 +(1-(20/20))*1=8,66	40
Paypal	/	32
S3	R(S3)=(60/85)*3+(50/90)*3 +(60/60)*3+(1-(30/30))*1 =6.78	100
S4	R(S4)=(85/85)*3+(90/90)*3 +(50/60)*3+(1-(5/30))*1 =9.33	59
Amazon	/	26

The service composition research domain has been subject of several surveys explaining methodologies, approaches and composition languages (Milanovic, 2004), (Srivastava, 2003).

In our methodology the service composition phase combines the selected and the developed services. The composition of services is a continuous process, which does not stop even after the deployment of the business application. A VCS (Version Control System) will manage different versions of compositions for each business application in order to allow backtracking, if necessary. If a new service S1' is discovered, with the same functionalities as S1 and better QoS, S1' replace S1 and a new composition with S1' instead of S1 is made, then a new version of the business application is generated.

3.4 IaaS Selection for Application Deployment

A Cloud infrastructure is either chosen by business stakeholder or automatically selected for the deployment of the business application, according to performance indicators required by the user and the QoS parameters characterizing the Cloud Infrastructure. IaaS selection is performed by ranking different IaaS, according to QoS indicators (response time, availability, accessibility, security, etc.) and following the ranking and selection processes described in section 3.3.

3.5 Development of Undiscovered Services

In traditional systems of service discovery, the discovery process will output the discovered business services, and gives no output if no services are discovered. In our methodology, when the service discovery algorithm does not return a result, we propose a development of the service. Undiscovered services are developed from UML (Unified Modelling Language) models. This phase identifies and develops the features described in the stakeholder's business application requirements that have not been discovered as business services. This involves three key steps:

1. *Undiscovered Services Modeling*. Services are modeled in UML without imposing a particular platform, in order to allow automatic generation of their code for a targeted Cloud platform with full knowledge of the tools it offers. Our work will reuse and integrate the CPIM library proposed by Giove and colleagues (Giove et al, 2013) for developing the undiscovered services.
2. *Discovery and Selection of Development Platforms (PaaS) for each service*. This phase allows to select the platform based on the service to be developed and the technologies allowed by the PaaS, as well as on the QoS requirements. This creates a distributed multi-platform development. The selection of platforms is the same as for the selection of infrastructure, with the only difference that we consider the APIs offered by different providers. The benefits of choosing a PaaS for each service development are: (i) the development is custom made to each service, (ii) dependencies of a particular PaaS are reduced, because a specific PaaS deployment is based on a MDD approach, (iii) addressing the distributed nature of the Cloud, thus increasing the availability of the final application.
3. *Automatic Deployment and Publication of New Developed Services*. After the development of new undiscovered services, these are deployed and published in a marketplace of services.

3.6 Automatic Deployment

The automatic deployment phase consists in automatically deploying the resulted business application on the IaaS preferred by the business stakeholder or on the one automatically selected by our system according to stakeholder's preferences and QoS parameters.

The deployment occurs until achieving stakeholder satisfaction. If after the tests and validation

phase, tests are not conclusive and/or the stakeholder does not validate results of the composition, IaaS resources are freed, and another composition is selected and deployed until stakeholder's satisfaction is achieved. Or the stakeholder can proceed to the refinement of its requirements.

3.7 Tests and Validation

The acceptance tests phase occurs after the business application deployment, where the stakeholder tests the deployed business application and verifies application features and the respect of business constraints, with the aim of validating or bringing modifications to the requirements expression.

If tests are conclusive, the stakeholder validates the deployed application. Else, other services are selected from the service marketplace, composed and deployed, or, the stakeholder introduces changes in his/her requirements expression for the business application, after, he/she has to test and validate the new deployed business application.

In the next section, we describe the architecture for ASDaaS. This architecture applies the previously explained methodology.

4 ARCHITECTURE FOR AUTOMATIC SOFTWARE DEVELOPMENT AS A SERVICE

The figure 4 illustrates the architecture for business application development as a service, which consists of three levels: client level, system level and service level. The client level allows to describe the application requirements (business functions, business constraints, QoS parameters and stakeholder preferences) via a web form; and to access to the deployed business application via internet.

The system level describes the core of the architecture and is the focus of the remainder of this section. It consists of the following services: project management, discovery as a service, PaaS discovery as a service, IaaS discovery as a service, undiscovered services development as a service, composition as a service and automatic deployment as a service.

The project management service is in charge of managing several projects. The discovery as a service is responsible for the service discovery operation based on service functions, QoS parameters and stakeholder preferences. The undiscovered services development as a service provides development of undiscovered services from models, on the Cloud

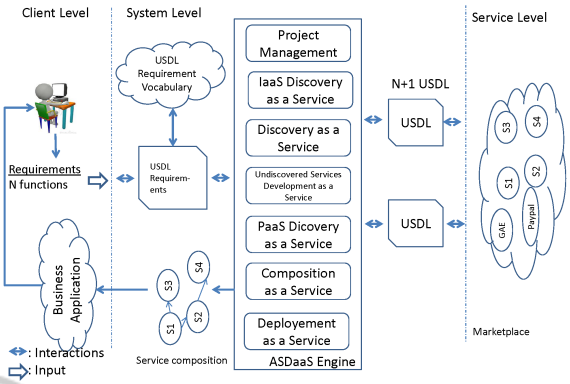

Figure 4: Automatic Software Development as a Service Architecture.

platform selected by the service PaaS discovery as service. This module generates the code exploiting the PaaS APIs and respecting the Cloud provider architecture. The developed services are then deployed. The "PaaS Discovery as a Service" module enables to find PaaS platform where each new developed service will be deployed. The IaaS discovery as a service enables the choice of an IaaS infrastructure for the deployment of the business application. This phase needs to match the QoS characteristics of different IaaS with those desired by the stakeholder. The deployment as a service consists of automatically deploying the generated business application on a selected infrastructure.

The service level includes the marketplace in which our system selects services for composition.

5 IMPLEMENTATION & VALIDATION

In order to validate and evaluate our proposal, we are implementing a prototype for ASDaaS using Java language. The phases of requirements expression, services discovery and IaaS selection are implemented. We use the Apache Jena Library for modeling and manipulating the USDL files and the SPARQL Query Language for querying them. We have implemented an IT history service that is responsible for returning the QoS parameters values for each discovered service.

Due to the early stage of implementation, few tests have been done. We have evaluated the service discovery execution time variation with the number of services in the marketplace. Figure 5 depicts these results obtained for the service discovery phase for our running scenario.

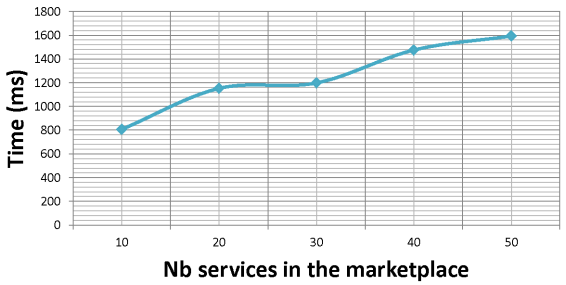

Figure 5: Variation of the service discovery execution time versus the marketplace's number of services.

6 CONCLUSIONS

With Cloud computing, the application development is changing, providing environments enabling to empower business stakeholders and to automatize the software development activities. In this paper, we have described a methodology for Cloud-based collaborative software development, and then presented the ASDaaS architecture that addresses the challenges of interoperability, independency to a specific platform, and respects the distributed nature of the Cloud environment. ASDaaS generates a business application based on discovered and developed on-the-fly business Cloud services.

Currently we are implementing the business Cloud service composition module. In the future, we will deploy the ASDaaS architecture in the Cloud Platform of the Lyon 1 University for evaluating and testing this work with a large set of cloud services.

REFERENCES

Amazon. (2013). Available: http://aws.amazon.com/fr/ec2/

Ardagna, Danilo., Di Nitto, Elisabetta., Casale, Giuliano., Petcu, Dana., Mohagheghi, Parastoo., Mosser, Sbastien., Matthews, Peter., Gericke, Anke., Ballagny, Cyril., DAndria, Francesco., Nechifor, Cosmin-Septimiu., and Sheridan, Craig. (2012). MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. *ICSE Workshop on Modeling in Software Engineering* (MISE), 2012. pp 50-56.

Cardoso, J. (2013). A Unified Language For Service Description: A Brief Overview. Available: http://www.issip.org/2013/04/26/a-unified-language-for-service-description-a-brief-overview/

Cardoso, J. B. (2010). Towards a unified service description language for the internet of services: Requirements and first developments. *IEEE International Conference on Services Computing, Florida*, USA (2010).pp 602 - 609.

Cardoso, J., Pedrinaci, C., Leidig, T., & Rupino, P. a. (2012). Open semantic service networks. International Symposium on Services Science 2012 (ISSS 2012). pp 141-154.

Coulouris, G. D., Dollimore, J., Kindberg, T., and Blair, G., (2011). Distributed Systems: Concepts and Design. Fifth Edition, published by Addison Wesley, May 2011.

Figueira, J.; Greco, S. and Ehrgott M.,. (2005). Multiple Criteria Decision Analysis: State of the Art Surveys. *Springer, 2005*.

Giove, Filippo., Longoni, Davide., Shokrolahi Yancheshmeh, Majid., Ardagna, Danilo. and Di Nitto, Elisabetta. (2013). An Approach for the Development of Portable Applications on PaaS Clouds. *CLOSER 2013 - 3rd International Conference on Cloud Computing and Services Science*, 2013. pp 591- 601.

Hepp M. (2011). GoodRelations Language Reference. Available: http://www. hepp-netz.de/ontologies/goodrelations/v1.html

Guha, Radha. and Al-Dabass, David. (2010). Impact of Web 2.0 and Cloud Computing Platform on Software Engineering. *IEEE, International Symposium on Electronic System Design,* 2010. pp 213-218.

Kommalapati, Hanu., and Zack, William H. (2011). The SaaS Development Lifecycle. 2011. Available: http://www.infoq.com/articles/SaaS-Lifecycle.

Linked Data - Connect Distributed Data across the Web. (2013). Available: http://www.linkeddata.org/

Linked Open Vocabularies (LOV). (2013). Available: http://lov.okfn.org/dataset/lov/

Linked USDL. (2013). Available: http://www.linked-usdl.org/

Milanovic, N. a. (2004). Current Solutions for Web Service Composition. *IEEE Internet Computing.* Vol 8. pp 51-59.

Linked USDL modules. (2013). Available: https://github.com/linked-usdl.

MODACLOUDS. Available: http://www.modaClouds.eu/.

Sledziewski, Krzysztof., Bordbar, Behzad., and Anane, Rachid. (2010). A DSL-based Approach to Software Development and Deployment on Cloud. *24th IEEE International Conference on Advanced Information Networking and Applications,* 2010. pp 414-421.

Srivastava, B. a., Koehler, J., (2003). Web service composition-current solutions and open problems. *Workshop on Planning for Web Services*. page 28-35.

Sun, Hailong., Wang, Xu., Zhou, Chao., Huang, Zicheng., Liu, Xudong.,. (2010). Early Experience of Building a Cloud Platform for Service Oriented Software Development. *2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*. pp 1-4.