On the Role of Ontologies in the Design of Service Based Cloud Applications

Fotis Gonidis[1], Iraklis Paraskakis[1], and Anthony J.H. Simons[2]

[1] South-East European Research Centre (SEERC),
City College – International Faculty of the University of Sheffield,
24 Proxenou Koromila Street, 54622 Thessaloniki, Greece
{fgonidis,iparaskakis}@seerc.org
[2] Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street,
Sheffield S1 4DP, United Kingdom
A.Simons@dcs.shef.ac.uk

Abstract. The wide exploitation of cloud resources has been hindered by the diversity on the provision of these resources and thus resulting in heterogeneity between them. Research efforts on the design of cloud applications, leveraging resources form heterogeneous cloud environments, have been concentrated on traditional cloud platform resources such as deployment capabilities and data stores. However, the emergence of the cloud application platforms has made available a wide range of platform basic services (e.g. e-mail, message queue and authentication service) that can drastically decrease the application development time. Our work focuses on eliminating the heterogeneity among the providers offering those services. To this end we propose an ontology-driven framework, which facilitates the seamless and transparent use of platform basic services provisioned by multiple clouds environments. Ontologies are leveraged to enable the homogeneous description of the functionality of the service providers.

Keywords: Multi-Cloud, Ontologies, Cloud platform service description.

1 Introduction

Cloud application platforms [1] are becoming increasingly popular and have the potential to change the way applications are developed, involving compositions of platform basic services. A platform basic service, in the Platform as a Service level (PaaS), can be considered as a piece of software which offers certain functionality and is reusable. Examples of such services are authentication mechanisms, logging mechanisms, message queues and email service. Such services are considered to be interwoven in the creation of many applications running from a cloud application platform and thus using the service instead of creating the corresponding code is of great benefit to the application developer. A service can be offered natively by the platform, such as the e-mail service offered by Google App Engine [2] and Amazon Elastic Beanstalk [3]. Alternatively, Independent Software Vendors (ISVs) can offer added-value services for a given platform, such as Heroku [4].

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part II, LNCS 8806, pp. 1–12, 2014.

The cloud application platforms have the potential to lead to a new paradigm of designing service-based cloud applications. Applications rather than being developed from the ground up, they can be synthesised from services offered by multiple clouds. This way developers can drastically increase their productivity and significantly shorten the time to market of the product.

However, an impediment for the wide exploitation of the available platform basic services constitutes the heterogeneity among the offered solutions. The heterogeneity mainly arises from (i) the variability in the workflow required to complete an operation [5] and (ii) the differences in the web API through which the service providers provision their services. This paper focuses on the latter variability point. For developers to leverage the full capabilities of services provided by multiple platforms should not be forced to develop an application directly against proprietary APIs, but rather should use either (i) standard and widely adopted technologies; or (ii) abstraction layers which decouple standard end-user APIs from the platform-specific APIs. To this end, this paper proposes an ontology-driven framework, which promotes the uniform access to platform basic services via the use of an abstract reference cloud API.

The use of ontologies is primarily motivated by their ability to support separation of concerns [6], that is enable the development of an application where the logic is separated from the data upon which it operates. This allows for data to be altered as much as it is required without altering the code related to the logic that operates on the data. In our case the data are the descriptions of the platform basic services that could be consumed by the various applications. Therefore, future service providers can be supported on the fly through an ontological description of their service. The framework is capable of reading the description and generating automatically the provider specific source code.

The rest of the paper is organised as follows. The next Section attempts to contextualise the scope of the proposed framework by defining the cluster of cloud platforms that it will focus on. Established work on the field related to the proposed solution is reviewed in Section 3. Thereafter, the main components of the framework are described namely, the ontologies and the core engine, which is responsible for generating the provider specific code.

2 Clustering of Cloud Platforms

Before stepping into the details of the proposed solution, the application scope of the approach needs to be defined. Particularly, we attempt a high-level clustering of the cloud platforms environments and subsequently we state the focus of our research. The clustering of the cloud platforms has been primarily based upon the adopted technologies and the provisioning of additional platform services either natively or via a service marketplace. From earlier surveys and reports [7], [8], [9], [10], [11], [12], [13] we find that cloud platform solutions can be clustered into three broad categories:

The first category includes platforms, which adopt standard and widely used technologies, such as popular programming languages and databases. They provide

basic development resources only, such as an application server and a database, and do not offer further cloud platform services or a service marketplace. An example platform in this category is CloudBees [14].

The second category includes platforms, which offer additional services via APIs such as e-mail service, image manipulation service and a message queue service. The services are offered natively by the platform such as the Google App Engine [2]. Alternatively the platforms may offer additional services via a marketplace such as the Heroku [4] add-ons.

The third category includes platforms, which adopt a native application development paradigm, where developers are expected to use bespoke visual tools and graphical interfaces to create the applications. Additional services can be offered by ISVs via marketplaces. However, those services are tightly integrated to the platform and no programming library or web interface is exposed. Platforms in this category include Zoho Creator [15].

Regarding the provisioning of services, platforms in the first category offers only deployment capabilities without any additional services. On the other edge of the spectrum, platforms in the third category are characterised by proprietary development tools and technologies. The lack of programming APIs makes them intractable when it comes to abstracting the offered services. Consequently, this paper focuses on platforms in the second category. Specifically, we are interested in the proprietary APIs that the services expose and the way these APIs can be abstracted in order to enable uniform and transparent access to the services.

3 Related Work

The constant increase in the offering of platform basic services has resulted in a growing interest in leveraging services from multiple clouds. Significant work has been carried out on the field, which can be grouped into three high-level categories: middleware platforms, Model-driven Engineering techniques and library based solutions. Representative work on each of the three categories is listed.

Library-based solutions such as jclouds [16] and LibCloud [17] provide an abstraction layer for accessing specific cloud resources such as compute, storage and message queue. While, library-based approaches efficiently abstract those resources, they have a limited application scope which makes it difficult to reuse them for accommodating additional services.

Middleware platforms constitute middle layers, which decouple applications from directly being exposed to proprietary technologies and deployed on specific platforms. Rather, cloud applications are deployed and managed by the middleware platform, which has the capacity to exploit multiple cloud platform environments. mOSAIC [18] is such a PaaS solution which facilitates the design and execution of scalable component-based applications in a multi-cloud environment. mOSAIC offers an open source API in order to enable the applications to use common cloud resources offered by the target environment such as virtual machines, key value stores and message queues.

Initiatives that leverage MDE techniques present meta-models, which can be used for the creation of cloud platform independent applications. The notion in this case is that cloud applications are designed in a platform independent manner and specific technologies are only infused in the models at the last stage of the development. MODAClouds [19] and PaaSage [20] aim at cross-deployment of cloud applications. Additionally, they offer monitoring and quality assurance capabilities. They are based on CloudML, a modelling language which provides the building blocks for creating applications deployable in multiple IaaS and PaaS environments. Hamdaqa et al. [21] have proposed a reference model for developing applications which leverage the elasticity capability of the cloud infrastructure. Cloud applications are composed of CloudTasks which provide compute, storage, communication and management capabilities. MULTICLAPP [22] is a framework leveraging MDE techniques during the software development process. Cloud artefacts are the main components that the application consists of. A transformation mechanism is used to generate the platform specific project structure and map the cloud artefacts onto the target platform. Additional adapters are generated each time to map the application`s API to the respective platform`s resources.

The solutions listed in this Section focus mainly on eliminating the technical restrictions that each platform imposes, enabling this way multi-cloud deployment of applications. Additionally, they offer monitoring and quality assurance capabilities as well as the creation of elastic applications. On the contrary, the vision of the authors is to facilitate the use of platform services, such as e-mail service, authentication service etc. and concrete providers from heterogeneous clouds in a seamless manner. To this end, we envision the creation of a framework, which enables the uniform description of the API of the services and the concrete providers. In turn, this will facilitate the design of applications, which leverage services from multiple cloud application platforms without being bound to the specific proprietary APIs.

4 Ontology Driven Framework

Towards enabling the design of service-based cloud applications, we present our solution approach, which is based on an ontology driven framework. With respect to the classification of the cloud platforms performed in Section 2, the proposed solution targets the platforms in the second category, namely the ones who offer platform services either natively or via a marketplace through a web API. The framework receives a description of the service functionality and subsequently generates automatically the client adapter to map onto the abstract reference API for the specific service.

In order to evaluate the effectiveness of the framework we apply the solution approach to the cloud e-mail service. The e-mail service allows a cloud application to send and receive e-mails without the need for the developer to set up and maintain an e-mailing server. Instead the service is offered by the cloud provider via a web interface.

The choice of the e-mail service was motivated primarily by the need for enabling cloud applications to leverage services from multiple clouds. The emergence of the cloud application platforms and the service marketplaces has made available a wide range of services which the cloud application should be capable of exploiting. To this end, the role of ontologies is explored as enablers for vendor specific API abstraction. The concrete provider`s API is captured in an ontology and subsequently is mapped to the abstract reference API which is exposed to the cloud developers.

4.1 Benefits of Using Ontologies

Ontologies are the novel aspect of the framework. They are used in order to allow a uniform description of the platform basic services. According to Gruber [23], they are formal knowledge over a shared domain that is standardized or commonly accepted by certain group of people.

The advantages here are two-fold. First, ontologies allow to define clearly the domain model of our interest; in our case the domain model is the cloud platform services offered by multiple platforms. The fact that an ontology is shared and commonly accepted description of a service, contributes towards the homogenisation of the latter. The cloud vendors can adhere to and publish the description of their service based on the common and shared ontology.

Moreover, ontologies can be reused and expanded if necessary. Thus, an ontology describing a platform basic service may not be constructed from the ground up but may be based on an existing one such as USDL (Unified Service Description Language) [24].

The reasoning capabilities that ontologies offer may be exploited for consistency check of the service descriptions.

Furthermore, mature tools are available in order to create and manipulate an ontology. Specifically, Protégé [25] is a well-established tool that allows users to create and edit ontologies, whereas the OWL API [26] and Jena framework [27] are among the popular Java frameworks that enable developers to manipulate ontologies using the Java language.

It is not the first time that ontologies are used in the cloud computing domain to enable service description. mOSAIC [28] ontology is used to enable service discovery and brokerage. This further motivates our choice of using ontologies for enabling service description. However, while the mOSAIC ontology focuses on describing the general and quality characteristics of a service, our ontology aims at the concrete functionality and the API of the services.

4.2 Architecture of the Ontology Driven Framework

Figure 1 depicts the high-level overview of the abstraction framework. The developer initiates the development of the application using a popular development environment such as Eclipse and a programming language such as Java. When the application

requires a platform basic service that is supported by the framework, the API description of the service is inserted into the framework. Consequently, the service description is parsed and the source code for the particular service is generated. The proposed cloud abstraction framework consists of two main parts: the models that represent the supported platform basic services and the core engine of the framework.

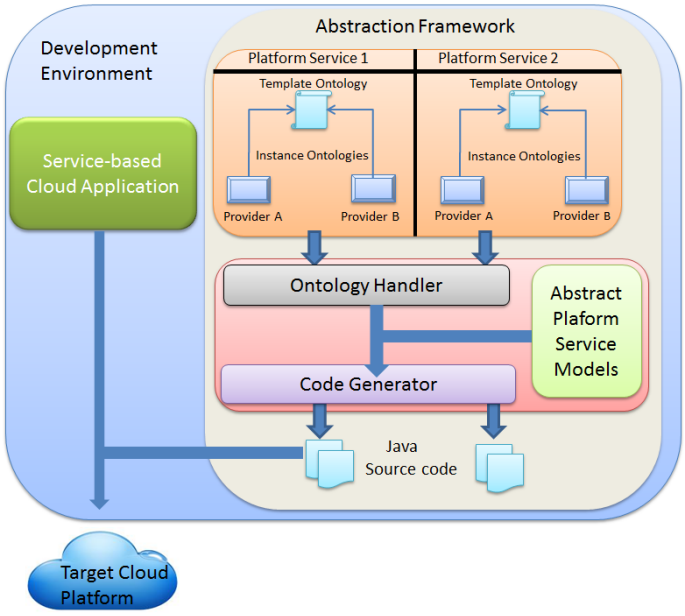

Fig. 1. High-level overview of the ontology-driven framework

Platform Service Models. The models represent the services that the framework supports. As mentioned in the previous Section, ontologies are used in order to build the models. The models, as seen in Figure 2 are structured into three levels. Inspiration has been gained by the Meta-Object-Facility (MOF) standard [29] defined for the Model Driven Engineering domain. Specifically, the hierarchy of the ontologies resembles the bottom three levels of the MOF structure, namely the meta-models, the models and the instances of the models.

The level 2 Ontology (O2) includes the description of the abstract platform services. Common concepts that define the platform basic services are captured at this level. Information about the configuration settings and the authentication mechanisms of the service are included. The O2 level also contains concepts, such as Operations and Attributes, required to describe an API.

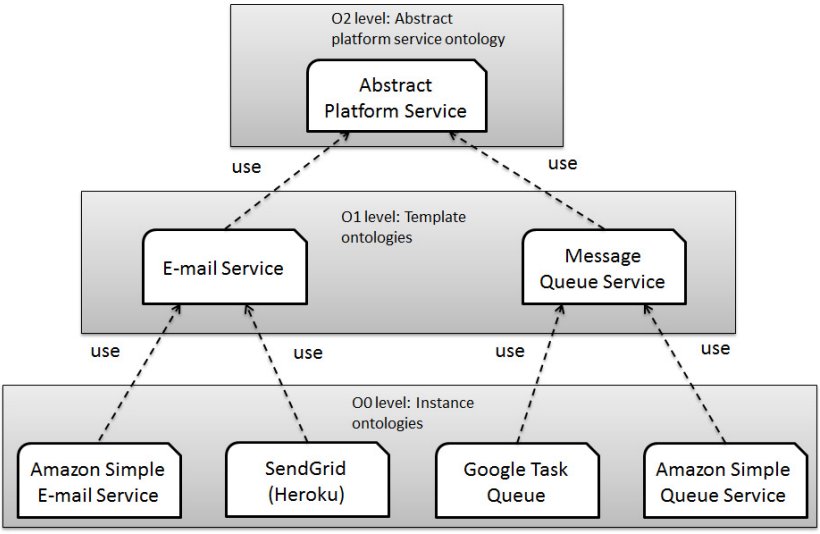

Fig. 2. The three levels of the ontology hierarchy

The level 1 Ontologies (O1) includes the concrete description of each of the platform basic services, which are supported by the framework. A dedicated ontology corresponds to each of the services and captures information about the functionality that they expose. For example, in the case of the cloud e-mail service, information that is captured in the O1 ontology describes the functionality for performing actions related to sending, receiving and manipulating e-mails. The ontologies in the O1 level are also referred to as Template ontologies.

The level 0 Ontologies (O0) include the description of the specific platform service providers. A dedicated ontology corresponds to each of the service providers and describes the native vendor specific API. For example, in the case of the e-mail service, an O0 ontology describes the concrete operations and attributes that a provider specific API exposes. The ontologies in the O0 level are also referred to as Instance ontologies. The users of the framework can form the Instance ontologies after reading the service providers` API. Alternatively, the Instance ontologies are created and published by the service providers and are automatically discovered by the framework.

In order to further clarify the use of the three levels of ontologies and the relationships among them, a simple description of a vendor specific API is constructed. The example that follows serves only illustrative purposes. Therefore, for the sake of simplicity only the necessary amount of information has been included. The example focuses on the cloud e-mail service and particularly on the description of the operation, which allows the users to send e-mails.

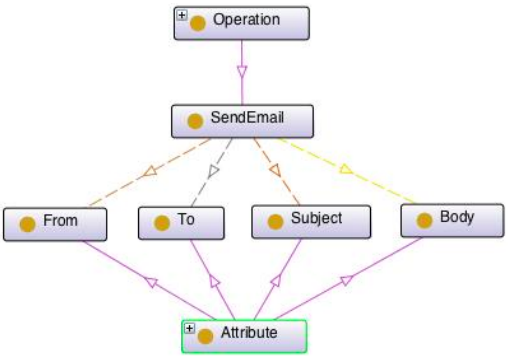

Fig. 3. Example of Template ontology for the cloud e-mail service

Figure 3 shows a snapshot of the Template ontology for the e-mail service, which describes the operation for sending an e-mail. The name of the operation is "SendEmail". It is a subclass of the class "Operation". "Operation" is defined in the Abstract Platform Service Ontology (O2 level) and includes all the operations offered by the service. Figure 3 also includes the following four elements: "From", "To" "Subject" and "Body". "From" denotes the sender of the e-mail, while "To" refers to the recipient. "Subject" refers to the title of the e-mail and "Body" holds the content. All four elements are subclasses of the class "Attribute". The class "Attribute" is defined in the Abstract Platform Service Ontology and includes all the attributes that are used for the execution of the operations. An attribute is linked to a specific operation with a property. Specifically, the four afore mentioned attributes are linked to the "SendEmail" operation with the following properties respectively: "hasFrom", "hasTo", "hasSubject" and "hasBody".

Figures 4 and 5 show how the service providers can describe their service and their specific API by creating their own ontology (Instance ontology), which is based on the publicly available Template ontology (Figure 3). Particularly, two Instance ontologies are shown which correspond to the Amazon Simple E-mail Service (SES) [30] and to SendGrid [31], a cloud E-mail service offered as add-on via Heroku.

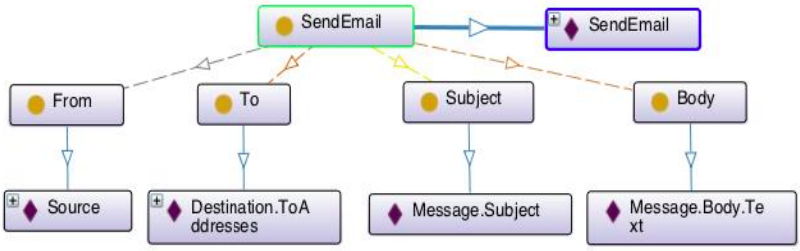

Fig. 4. Example of Instance ontology for the Amazon SES service

Figure 4 shows the parameters required to send an e-mail as defined in the API of the Amazon SES service. Individuals are created to express each of the specific

elements of the provider`s API. An individual, in ontologies, can be considered a member of a class. Specifically, the “send” individual denotes the operation name, which is equivalent to the “SendEmail” operation of the Template ontology. This justifies the fact that “send” individual is of type “SendEmail”. The individual “Source” denotes the sender of the e-mail and is equivalent to the “From” attribute. Thus, it is defined of type “From”. Likewise the individual “Destination.ToAddresses” is of type “To”, the “Message.Subject” is of type “Subject” and the “Message.Body.Text”, which holds the content of the e-mail, is of type “Body”.

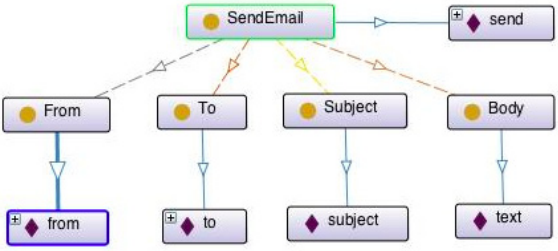

Fig. 5. Example of Instance ontology for the SendGrid (Heroku)

In the same way an Instance ontology can be created (Figure 5) to describe the API of the “SendGrid” E-mail service provider offered via Heroku. The individual “SendEmail” identifies the operation of sending an e-mail and is equivalent to the “SendEmail” of the Template ontology. Therefore it is of type “SendEmail”. Likewise, the individuals “from”, ”to”, “subject” and “text” are of type “From”, “To”, “Subject” and “Body” respectively.

In the same way the rest of the functionality of a platform basic service can be described. At the same time, the differences in the APIs between the various providers can be captured. The e-mail service has been used as an example. The proposed structure of the three levels of ontologies can be used to describe any platform basic services, offered natively by the platforms or via marketplaces through a web API, such as message queue service, authentication service and payment service. Initially, a Template ontology is required to describe the functionality of each of the services. Consequently the Instance ontologies are created to capture the vendor specific APIs.

Core Engine of the Framework. The second part of the cloud abstraction framework comprises the core engine. It receives as input the service description of the platform basic services and generates the source code for the target cloud provider. As seen from Figure 1, the core engine can further be decomposed into the following three high–level components: The Ontology Handler, the Abstract Platform Service Models and the Code Generator.

The Ontology Handler is responsible for manipulating the ontologies which contain a platform service description. It first discovers the published Instance ontologies, which describe a particular service provider. Alternatively, the Instance

ontologies can be fed manually to the Ontology Handler. Then, it performs a consistency check to reassure that the Instance ontology conforms to the respective Template ontology. For that reason the capabilities of reasoning in ontologies may be exploited. Consequently, the ontology is parsed and an object representation is created in memory. The object representation of the ontology is then forwarded to the Code Generator.

The Abstract Platform Service models contain a collection of abstract models, which correspond to the Template ontology and describe the service. They can be considered as the scaffold of the service. They are later enriched with the provider`s specific API information.

The Code Generator, as the name implies, is responsible for generating the concrete source code for the target service provider. It receives as input the object representation of the parsed Instance ontology and the abstract service models. Then it enriches the abstract models with the provider specific information and outputs the provider specific source code. Thus, while the developer uses a single common API to access a platform service, internally the source code is adjusted to map each time to the concrete service provider API.

In order to illustrate the functionality of the proposed framework, the example of the cloud e-mail service has been used. However, the generic nature of the framework enables the support of any platform basic service offered via a web API. As described in Section 4, the Template and Instance ontologies are required. The former captures the reference functionality that is exposed to the developers. The latter includes the provider specific API and the mapping to the reference functionality.

The proposed solution is capable of abstracting efficiently the heterogeneities among the cloud providers` APIs and thus eliminating the exposure of the application to proprietary APIs. However, the approach is inherently limited to the abstraction of the common functionality offered by the cloud providers. This means that specific functionality that is provided only by one vendor is not included in the Template ontologies and therefore not mapped to the abstract reference API. In order to allow developers to use the provider specific functionality, the latter is described directly in the Instance ontologies. Then, additional client adapters can be generated and used by the developers. In case the functionality is adopted by additional providers it can be also included in the Template ontology. Therefore, the proposed framework rather than being static, it is continuously updated to accommodate new features offered by the platform basic services.

5 Conclusions

In this paper, we addressed the issue of the design of service-based cloud applications capable of leveraging services offered by multiple cloud environments. To this end, we presented an ontology driven framework, which facilitates: (i) the description of the functionality of concrete service providers, (ii) the provisioning of a common platform service API to be used independently of the target provider and (iii) the automatic generation of the client adapters required to consume the target services. The proposed solution comprises two main parts: the ontological description of the

services and the core engine. The first one includes the Template and Instance ontologies, which contain the abstract and the provider specific service description respectively. The second part reads the Instance ontology and generates the source code for the target provider. Thus, the use of the ontology driven framework facilitates the design of applications exploiting services from multiple platforms using provider independent API rather than being bound to proprietary technologies.

Acknowledgment. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement nº264840, the RELATE project (http://www.relate-itn.eu).

References

1. Kourtesis, D., Bratanis, K., Bibikas, D., Paraskakis, I.: Software Co-development in the Era of Cloud Application Platforms and Ecosystems: The Case of CAST. In: Camarinha-Matos, L.M., Xu, L., Afsarmanesh, H. (eds.) Collaborative Networks in the Internet of Services. IFIP AICT, vol. 380, pp. 196–204. Springer, Heidelberg (2012)
2. Google App Engine (2014), `https://developers.google.com/appengine`
3. Amazon Elastic Beanstalk (2014), `http://aws.amazon.com/elasticbeanstalk/`
4. Heroku (2014), `http://heroku.com`
5. Gonidis, F., Paraskakis, I., Simons, A.J.H.: A Development Framework Enabling the Design of Service-Based Cloud Applications. In: 2nd International Workshop on Cloud Service Brokerage. Springer, Manchester (2004) (in press)
6. Kourtesis, D., Paraskakis, I., Simons, A.J.H.: Policy-driven governance in cloud application platforms: an ontology-based approach. In: Proceedings of the 4th International Workshop on Ontology-Driven Information Systems Engineering, Graz (2012)
7. Badger, L., Grance, T., Patt-Corner, R., Voas, J.: NIST Cloud Computing Synopsis and Recommendations. Technical Report, National Institute of Standards and Technology (2012)
8. Kourtesis, D., Bratanis, K., Bibikas, D., Paraskakis, I.: Software Co-development in the Era of Cloud Application Platforms and Ecosystems: The Case of CAST. In: Camarinha-Matos, L.M., Xu, L., Afsarmanesh, H. (eds.) Collaborative Networks in the Internet of Services. IFIP AICT, vol. 380, pp. 196–204. Springer, Heidelberg (2012)
9. Khan, N., Noraziah, A., Ismail, E.I., Deris, M.M., Herawan, T.: Cloud Computing: Analysis of Various Platforms. Int. J. E-Entrep. Innov. 3(2), 9 (2012)
10. Pastaki Rad, M., Sajedi Badashian, A., Meydanipour, G., Ashurzad Delcheh, M., Alipour, M., Afzali, H.: A Survey of Cloud Platforms and Their Future. In: Gervasi, O., Taniar, D., Murgante, B., Laganà, A., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2009, Part I. LNCS, vol. 5592, pp. 788–796. Springer, Heidelberg (2009)
11. Ried, S., Rymer, J.R.: The Forrester WaveTM: Platform- As-A-Service For Vendor Strategy Professionals, Q2 2011. Technical Report, Forrester (2011)
12. Development in the Cloud: A Framework for PaaS and ISV Flexibility, Saugatuck Technology Inc. (2010)

13. Gonidis, F., Paraskakis, I., Simons, A.J.H., Kourtesis, D.: Cloud Application Portability. An Initial View. In: 6th Balkan Conference in Informatics, pp. 275–282. ACM, Thessaloniki (2013)
14. CloudBees (2014), http://www.cloudbees.com
15. Zoho Creator (2014), http://www.zoho.com/creator
16. jclouds (2014), http://www.jclouds.org
17. Apache LibCloud (2014), https://libcloud.apache.org/index.html
18. Petcu, D.: Consuming Resources and Services from Multiple Clouds. Journal of Grid Computing 10723, 1–25 (2014)
19. Ardagna, D., Di Nitto, E., Casale, G., Petcu, D., Mohagheghi, P., Mosser, S., Matthews, P., Gericke, A., Ballagny, C., D'Andria, F., Nechifor, C.S., Sheridan, C.: MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. In: Workshop on Modeling in Software Engineering, pp. 50–56. IEEE, Zurich (2012)
20. Jeffery, K., Horn, G., Schubert, L.: A vision for better cloud applications. In: Proceedings of the 2013 International Workshop on Multi-Cloud Applications and Federated Clouds, pp. 7–12. ACM, Prague (2013)
21. Hamdaqa, M., Livogiannis, T., Tahvildari, L.: A reference model for developing cloud applications. In: 1st International Conference on Cloud Computing and Services Science, pp. 98–103. Noordwijkerhout (2011)
22. Guillén, J., Miranda, J., Murillo, J.M., Cana, C.: Developing migratable multicloud applications based on MDE and adaptation techniques. In: 2nd Nordic Symposium on Cloud Computing & Internet Technologies, pp. 30–37. ACM, Oslo (2013)
23. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 5(2), 199–220 (1993)
24. Pedrinaci, C., Cardoso, J., Leidig, T.: Linked USDL: A Vocabulary for Web-Scale Service Trading. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 68–82. Springer, Heidelberg (2014)
25. The Protégé Ontology Editor and Knowledge Acquisition System (2014), http://protege.stanford.edu/
26. OWL API (2014), http://owlapi.sourceforge.net/
27. Jena Framework (2013), http://jena.apache.org
28. Moscato, F., Aversa, R., Di Martino, B., Fortis, T., Munteanu, V.: An analysis of mOSAIC ontology for Cloud resources annotation. In: Federated Conference on Computer Science and Information Systems, pp. 973–980. IEEE, Szczecin (2011)
29. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. In: Workshop on Metamodeling for MDA, York, pp. 179–197 (2003)
30. Amazon Simple E-mail Service (2014), http://aws.amazon.com/ses/
31. SendGrid (2014), http://www.sendgrid.com