Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities

Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu

Department of Computer Science, University of York,
Deramore Lane, York YO10 5GH, UK
gabriel@cs.york.ac.uk,
{louis.rose,radu.calinescu}@york.ac.uk

Abstract. Different cloud platforms offer similar services with different characteristics, names, and functionalities. Therefore, describing cloud platform entities in such a way that they can be mapped to each other is critical to enable a smooth migration across platforms. In this paper, we present a DSL that uses a common cloud vocabulary for describing cloud entities covering a wide variety of cloud IaaS services. Through analysis of existing cloud DSLs, we advocate that our cloud DSL is more expressive for the purpose of describing different cloud IaaS services. In addition, when used along with TOSCA, our preliminary analysis suggests that our Cloud DSL significantly reduces the workload of creating cloud descriptions in a TOSCA specification.

Keywords: DSL; Cloud computing; Portability; TOSCA

1 Introduction

Costa Coffee, Starbucks, and Caffè Nero are the three largest coffee shop companies in the UK. Although they all sell their coffees in three common sizes – small, medium, and large – they name these sizes differently. This can create a lot of confusion. A Starbucks customer might get frustrated when ordering a *Grande* in Caffè Nero since that, in the Starbucks vocabulary *Grande* means medium, whereas for Caffè Nero it means large. Likewise, those unfamiliar with Starbucks might find contradictory that *Tall* is the smallest size. Furthermore, similar products also have different characteristics. For instance, a medium skimmed latte in Starbucks[1] contains 156% more calories than its Nero[2] version. Thus, although their products look similar, they are not all the same. Therefore, having a detailed description of each product is critical to prevent misunderstandings.

Like in the coffee-shop market, cloud platforms offer similar services, but with different names, characteristics, and functionalities. For example, consider the Amazon S3 and Dropbox storage services. Overall, they provide the same functionality: file storage, storage elasticity, and interfaces for management. However,

[1] http://www.starbucks.co.uk/quick-links/nutrition-info
[2] http://www.caffenero.co.uk/Nutrition/hotdrinks.aspx

a closer look reveals critical differences, including the use of different file systems. Whereas Dropbox has one single file system root, Amazon S3 uses multiple root containers called "buckets". Furthermore, a bucket has a region, which specifies a geographical location for the content stored within. Like for coffee shops, to support a smooth migration of applications across different cloud platforms, it is critical to describe the *semantics* of cloud entities.

Semantic differences are critical in cloud as they hinder the smooth migration of assets (e.g., data and applications) across providers [21]. For example, the differences between Amazon S3 and Dropbox hinder the migration (transfer) of files from Dropbox to Amazon S3, as it is necessary to create a bucket and assign it to a region. Cloud portability, i.e. the ability to migrate an asset deployed in one cloud to another [18], is one of biggest challenges in cloud computing, and it has been widely addressed by both academia and industry [17].

Providing homogeneous description of cloud platforms is a potential solution to overcome semantic differences and achieve cloud portability in this highly heterogeneous environment [3], [15], [19]. Recent research reveals three means to describe cloud platforms [22]. Although we present them here, discussing their benefits and drawbacks is beyond the scope of this paper.

(i) A *platform abstraction* solution consist of describing concepts of either the entire platform or its elements, at different levels, e.g. as proposed by MODA-Clouds [3]. This solution can adopt different technologies to achieve their goals, such as ontologies [6] and Model-Driven Architecture [3]. This solution covers both design- and run-time, e.g. as proposed by meta cloud [20];
(ii) *Standardized references* focus on design-time only through at defining references for cloud platforms [15], APIs [7], [1], [9], or applications [11]. In the context of our paper, a reference is a set of rules, or constraints that a cloud user or provider must follow. However, most of solutions in this type focus on setting up references for cloud APIs. According to Escalera & Chavez, cloud APIs consist of software libraries used by application developers to manage cloud services [7]. Apart from [11], which targeted cloud applications, other solutions in this type target only cloud providers; and
(iii) *Domain Specific Languages* (DSLs). A DSL is a language tailored for a particular domain or context [5]. Like *standardized references*, DSL-based solutions are for use at design-time. However, some solutions might provide support for run-time mechanisms, such as CloudMF [8]. As DSLs are defined for a particular purpose [16], these solutions cover a wide range of goals, such as automatic generation of mobile-cloud applications [19], and description and comparison of Service Level Agreements (SLAs) [2]. Regardless of the purpose, meta-models are the cornerstone of these solutions. Finally, DSL-based solutions are mainly intended to support cloud users.

In this paper, we present a DSL that uses a common cloud vocabulary for describing cloud platform entities, such as services and resources across a wide variety of cloud platforms. Unlike existing Cloud DSL, this work covers a wide variety of cloud IaaS services, contributes to different phases of cloud portability, facilitates the communication of services and resources to different levels of

stakeholders, and enables the description of different types of clouds, such as federation and inter-clouds. In addition to positioning this work in the related literature (Section 5), we contribute to the advance of the state-of-the-art in both cloud portability and Model-Driven Engineering (MDE) by: (i) supporting cloud portability via a common meta-model for cloud computing (Section 2); (ii) facilitating the visualization and communication of cloud assets amongst different stakeholders by providing a graphical editor (Section 3); and (iii) reducing the effort of describing cloud entities in TOSCA cloud standard [4] (Section 4).

2 Cloud Meta-model

The meta-model, which describes the domain covered by the language, is the cornerstone of a DSL. A DSL consists of abstract and concrete syntax — whereas the abstract syntax defines the constructs of the language, the concrete syntax defines the representation of these constructs [5], [16]. For example, the abstract syntax of the Web Service Description Language (WSDL) defines a set of entities and their properties, such as *ServiceType* and *InterfaceType*, representing, respectively, a service exposed to a client, and its interfaces. To specify *Services* and *Interfaces*, one uses XML statements like <*service* /> and <*interface* />. These XML statements are the concrete syntax of the WSDL.

Due to the heterogeneity of cloud platforms and services, creating a cloud meta-model that covers a broad range of cloud services is not a straightforward task. To devise a cloud meta-model, we: (i) analysed four sources of information; (ii) identified correspondences amongst similar entities across these different sources; and (iii) combined entities and their relationships into a new meta-model. Our analysis started with an extensive literature review [22], in which we identified some critical cloud entities. Next, we leveraged the contribution of two important standardization efforts, OGF OCCI [9] and DTMF CIMI [1]. We also investigated relevant research projects focusing on cloud portability, in particular MODAClouds[3] and REMICS[4]. Finally, we examined a wide range of cloud IaaS services, including Amazon SQS, Microsoft Azure Compute, and Rackspace Cloud Files. Figure 1 shows our cloud meta-model.

A cloud *Platform* provides *Service*, such as computing and storage. A cloud *Platform* has a name, such as Amazon Web Services, or OpenNebula. A *Service* might be managed through multiple *Management Interfaces*, which are provided by cloud *Platform*. A *Management Interface* has a type, such as RESTful, or Query-based, and properties, such as authentication attributes. The *Platform* is responsible for the *Cloud User* management. A *Cloud User* is identified by its name, and can have several keys to access its *Resources*. A *Service* is identified by its name, such as EC2, or Cloud Servers. A *Service* might be supported by other service. For example, Amazon EBS and S3 supports Amazon EC2, providing persistent storage. Each *Service* might operate in a different *Region*. A *Region* represents a wide geographical location, such as Europe or Asia. In addition to

[3] http://www.modaclouds.eu
[4] http://www.remics.eu

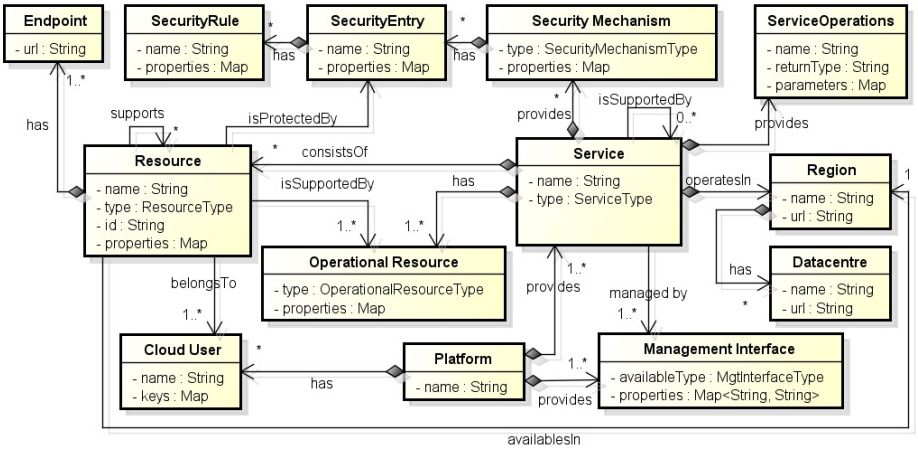

Fig. 1. Abstract syntax of the Cloud DSL

its name, a *Region* might have a particular code assigned by the cloud *Platform*. This code is represented in this meta-model by the *url* attribute. A *Region* might have several *Datacentres*, which are identified by their names and urls.

A *Service* has *ServiceOperations*. A *ServiceOperation* represents those management operations that a *Cloud User* can perform on its *Resource* through a *Management Interface*, such as sending a message to a queue in a message queue service. A *ServiceOperation* is identified by its name, such as runInstance or listImages, for example. Some operations rely on input parameters, such as the name of image, and region. As these operations are implemented by APIs, they have a single return value, which might be a list of results, for example. A *Service* exists to provide *Resources*, such as VMs and storage containers. Each *Platform* might define different properties and states for *Resource*s provided by its *Service*. However, two properties are present in most of *Resource*s: id and name. Once the *Resource* has been created, it is made accessible by one or more *Endpoints*. The *Resource* is available in one of the *Region* supported by the *Service*. One *Resource* might support another. This is the case of storage containers, which are used to support a set of files (*Resource*).

Some *Resources* rely on *OperationalResource*. An *OperationalResource* represents internal resources provided by the *Service*, such as the hardware of a VM, or the engine of a database service. In addition to the *Cloud User* credentials, some services provide further *Security Mechanism*, such as firewall and permissions. A *Security Mechanism* consists of a set of *Security Entry*. For example, a file in a storage service might have different permissions (*SecurityEntry*). Each *SecurityEntry* is assigned to the *Resource* it protects. More sophisticated mechanisms require further elements, represented in the meta-model by *Security Rule*. This is the case of security groups, provided by Amazon EC2.

3 Proof of Concept

To implement our Cloud DSL, we adopted Epsilon[5], a suite of languages and tools for model management operations, such as model transformation and analysis. The implementation process consisted of four steps: (i) creating the cloud meta-model using Emfatic, a textual notation for creating Ecore models; (ii) annotating the meta-model with EuGENia annotations. EuGENia is a tool that takes advantage of model transformation techniques to mitigate the complexity of GMF and EMF [13]; (iii) generating the graphical editor using the EuGENia tool; and (iv) adjusting graphical components, such as figures used to represent cloud entities. The graphical editor consists of three parts (Figure 2): (i) a canvas, in which cloud entities and their relationships are represented by graphical components; (ii) a palette, which presents the cloud meta-model entities; and (iii) the properties tab, which shows the properties of each selected entity.

To evaluate the expressiveness of our Cloud DSL, we used our Cloud DSL to describe the Amazon Web Application Hosting (AWAH) reference architecture[6]. The reference architecture consists of seven different Amazon services, which provides several resources for a Web application, such as storage and DNS routing. In order to implement this reference architecture, we hosted the Java PetStore[7] application using the services defined in the reference. Figure 2 shows the description of two services: Amazon CloudFront and Amazon S3. Amazon CloudFront is a content distribution service, which routes requests to the nearest content storage location.

In this implementation of AWAH reference architecture, we have only one distribution configured, which represents a *Resource* for this service (*Distribution_PetPictures*). As Amazon CloudFront does not store the content, it relies on a storage service, in this case, Amazon S3. The figure shows three resources provided by Amazon S3: *PetPictures*, *petstorestaticpages*, and *index.html*. Whereas the first two resources are buckets, the third is a file. The two buckets are protected by a *Security Entry*, defined using the *Security Mechanism* (PERMISSION). The location of *PetPictures* bucket is explicitly defined by a *Region*, represented by *US Standard* in this example.

The concrete syntax of our Cloud DSL represents a *ServiceOperation* as a container inside the *Service*. *ServiceOperation* might have several parameters. In this example, the operations described represent those required to operationalise the AWAH architecture, such as starting an instance. In Figure 2, Amazon S3 operations are hidden in the *Service* entity (note the "+" signal just below the service name). Figure 3 (a), shows the Amazon RDS service and the operations described: *launchDBInstance*, and *terminateDBInstance*. Whereas the first operation has six parameters, the second has only one. Parameters are represented by an "i" signal. This notation is used throughout our Cloud DSL to represent parameters. For example, Figure 3 (b) shows two *OperationalResources* (HARD-

[5] http://www.eclipse.org/epsilon/

[6] `http://aws.amazon.com/architecture/?nc1=f_cc`

[7] `http://www.oracle.com/technetwork/articles/javaee/petstore-137013.html`

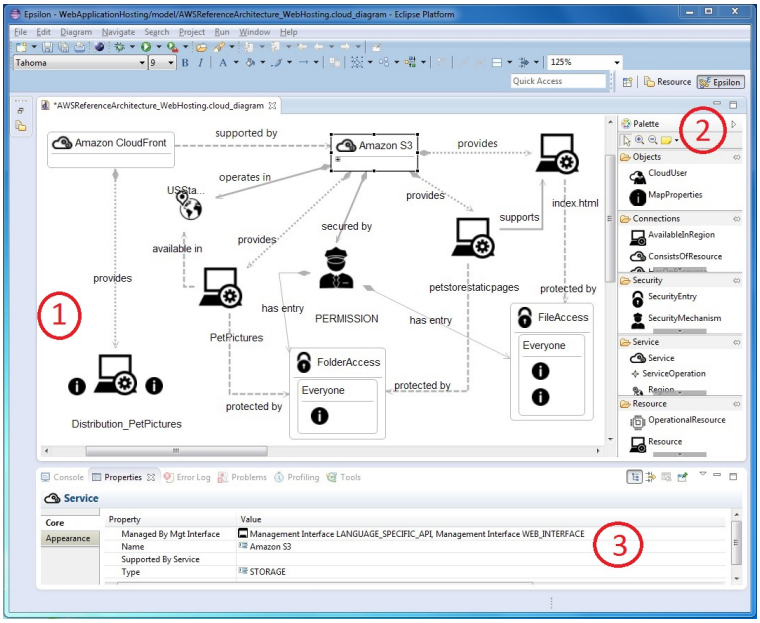

Fig. 2. The concrete syntax of the Cloud DSL implemented by a graphical editor

WARE and VM_IMAGE) which contain particular properties. Figure 3 (c) shows the properties tab for one cloud *Resource* (VM_INSTANCE).

Figure 3 (b) shows the Amazon EC2 service, and its related *Resource*, *DynamicWebSite*. This resource is a VM which hosts the dynamic content of the Java PetStore application. In the properties tab (Figure 3 (c)), it is possible to see the properties of this resource, such as id (generated by the cloud platform), and the *Cloud User* which owns the VM. *DynamicWebSite* is supported by two *OperationalResources*: HARDWARE, and VM_IMAGE. Whereas the former represents the type of instance used, the latter represents the Amazon Machine Image (AMI) used. The AMI contains the operational system as well as all applications required to run the Website. Different from Amazon S3 (Figure 2), Amazon EC2 enables specifying a particular datacentre. It is possible to note it in the Figure 3 (b), just above the globe, which represents a *Region*. Finally, in order to access the VM, an endpoint is made available. The concrete syntax for it is small rings, located just in the right side of *DynamicWebSite* resource.

4 Towards Simplifying Cloud Services and Resources Description in TOSCA

Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standard supported by OASIS, and intended to support application portabil-

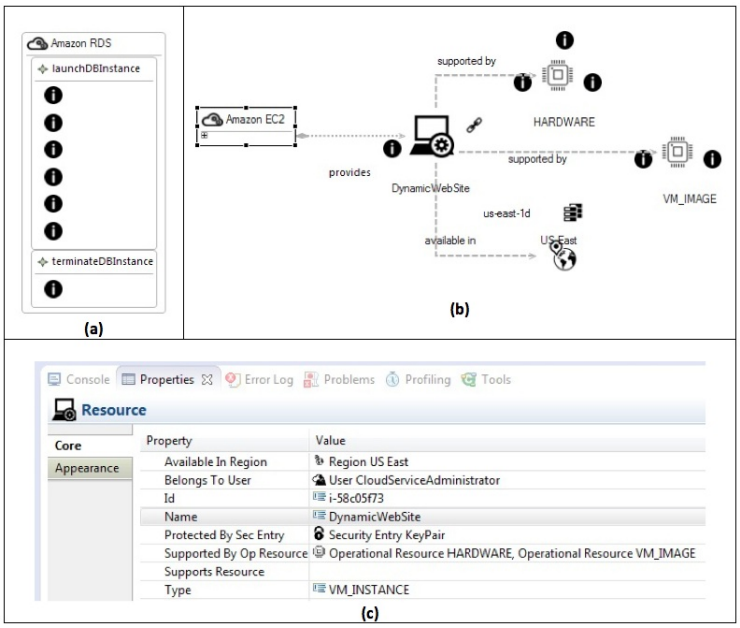

Fig. 3. Our Cloud DSL in action describing the AWAH reference architecture

ity across clouds. TOSCA defines types, that describe applications and cloud services, and templates that represent instances of these types. The TOSCA ecosystem comprises: specification and run-time environments. Whereas the former covers both application topology and activity orchestration, the latter is responsible for processing these specifications [4]. Several companies demonstrated the benefits of using TOSCA migrating an application across cloud platforms[8].

Despite from the benefits that TOSCA can provide, describing cloud resources in a TOSCA specification is a cumbersome task. TOSCA does not use the typical cloud vocabulary, such as services and resources. Instead, it defines a set of abstract elements, such as nodes, capabilities, and policies. Although this strategy enables the specification of both cloud and application components, it complicates the specification of cloud platform entities using TOSCA elements, specially because TOSCA official documentation does not define how to map cloud entities to TOSCA elements. In addition, as TOSCA specification is defined as a XML document, it is quite hard to have an overview of cloud entities.

Therefore, we proposed using our Cloud DSL to specify cloud services and resources for TOSCA specification. To this end, we are taking advantage of MDE techniques, in particular, model-to-model and model-to-text transformations (MT). As Hermans, Pinzger & van Deursen identified in their study [12], a DSL along with MT techniques contribute to reduce effort and increase productivity by automating repetitive tasks, such as code generation. Indeed, Brambilla,

[8] `https://www.oasisKopen.org/events/cloud/2013/TOSCAdemo`

Cabot & Wimmer analyse that code generation can save much effort for implementing CRUD operations, which are responsible for 80% of software functionality in data-intensive applications [5]. To achieve these benefits, we have begun work on mapping between our cloud meta-model and TOSCA elements.

However, this cloud-to-TOSCA mapping is an on-going work, which has required substantial intellectual and technical effort. Thus, describing these mappings is beyond the purposes of this paper. Here, we report on what we want to achieve once the mapping is complete. Our preliminary analysis has shown that it is possible to achieve similar results to those reported in [12], in particular, effort reduction. Our hypothesis for such statement is underpinned by the fact that a single cloud entity can be mapped to more than one TOSCA element.

For example, a cloud *Service* is represented in TOSCA as a *TNodeType*. However, as exists dependences between services, they also become a *TCapabilityType* and a *TRequirementType*. For instance, Amazon EC2 relies on Amazon EBS to store persistent data. Therefore, Amazon EBS provides the storage capability whereas Amazon EC2 requires such a capability. In addition, a cloud *Service* carries information used by TOSCA *TNodeTemplate*. Thus, writing a TOSCA specification manually would require that those four entities and all their related information were encoded by a human developer — which is both a time consuming and an error-prone activity.

5 Related Work

In 2010, Gonçalves et al. presented the first DSL devised specifically to describe cloud entities, CloudML [10]. CloudML is underpinned in the D-Cloud context aiming at allowing cloud computing providers to describe both cloud resources and services, and cloud developers, to describe their computing requirements. D-Clouds stands for Distributed Clouds, and authors define it as "*smaller datacentres sharing resources across geographic boundaries.*" CloudML is an XML-based language, and it is based on three requirements: (i) representation of physical and virtual resources as well as their state; (ii) representation of services provided; and (iii) representation of developer's requirements. In contrast to this DSL, our work is not limited to a particular context. As our Cloud DSL captures essential characteristics of cloud platforms, it can be used to model different types of clouds, such as federation and inter-cloud.

In 2011, Liu & Zic presented Cloud#, a textual DSL that enables cloud providers to describe internal organization of cloud resources [14]. Focused on computing services, their requirements are: (i) to express computation units and different privilege levels of computation; (ii) to allow programmable bidirectional control and data transfer between computation units; and (iii) to model physical resources. The two cornerstone entities in their meta-model are *CUnit*, which represents a cloud, a virtual machine, or an operating system; and *Action*, which defines a computation task. Different from this DSL, which defines a textual language to describe computing services, our Cloud DSL provides a graphical representation of cloud entities, presented in a diagram. Thus, our Cloud DSL

facilitate not only the visualisation of cloud entities, but also the communication of the cloud strategy amongst different levels of stakeholders.

In 2012, Alkandari & Paige reported on-going work towards a DSL for describing and comparing SLAs offered by different cloud providers [2]. Authors came up with two meta-models, one for describing SLAs offered by cloud provider, and another to describe SLAs required by cloud users. In addition, an algorithm was developed to compare models from cloud users to those from cloud providers. However, this DSL is limited to the first phase of cloud portability - analysis. Although our Cloud DSL cannot describe SLAs with the richness of detail as this DSL does, our Cloud DSL contributes to different phases of cloud portability, such as analysis and migration.

Finally, in 2013, Ferry et al. introduced the CloudMF [8]. CloudMF aims at supporting provisioning and deployment of applications in multiple clouds at run- and design-time. To accomplish this objective, CloudMF covers four requirements: (i) separation of concerns; (ii) provider independence; (iii) reusability; and (iv) abstraction. CloudMF consists of two components: (i) CloudML, the modelling environment (DSL); and (ii) Models@run-time, which provides an abstract representation of the running system. However, this DSL is limited to computing and storage services. Our Cloud DSL enables the description of a wide variety of cloud services, such as message queue, scaling, and DNS routing.

6 Conclusion

This paper introduced a Cloud DSL which supports cloud portability by describing cloud platform entities. The wide coverage of cloud IaaS services, and our ongoing work towards the integration of our Cloud DSL and TOSCA, suggest that our Cloud DSL can cover critical aspects of cloud service specification. As next step, we will investigate literature to identify means of evaluating and comparing our Cloud DSL to other cloud-related languages. Finally, in our project, we have been working towards mapping our Cloud DSL to platform-specific offerings by mapping entities of cloud meta-model to platform-specific cloud APIs. Exploiting these capabilities requires the maintenance of these mappings.

Acknowledgments. This work was funded in part by CNPq - Brazil and EU FP7 project OSSMETER (Contract #318736).

References

1. DTMF CIMI, `http://www.dmtf.org/standards/cloud`
2. Alkandari, F., Paige, R.F.: Modelling and comparing cloud computing service level agreements. In: 1st Intl Workshop on Model-Driven Engineering for High Performance and CLoud computing. pp. 1–6. ACM Press, New York, USA (2012)
3. Ardagna, D., Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.S., Petcu, D., Gericke, A., Sheridan,

C.: MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. In: 4th Intl Workshop on Modeling in Software Engineering. pp. 50–56. IEEE, Zurich (Jun 2012)
4. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services, pp. 527–549. Springer, New York (2014)
5. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers (2012)
6. Ejarque, J., Alvarez, J., Sirvent, R., Badia, R.M.: A Rule-based Approach for Infrastructure Providers' Interoperability. In: IEEE 3rd CloudCom. pp. 272–279. IEEE, Athens (Nov 2011)
7. Escalera, M.F.P., Chavez, M.A.L.: UML model of a standard API for cloud computing application development. In: 9th Intl Conf on Electrical Engineering, Computing Science and Automatic Control. pp. 1–8. IEEE, Mexico City (Sep 2012)
8. Ferry, N., Chauvel, F., Rossini, A., Morin, B., Solberg, A.: Managing multi-cloud systems with CloudMF. In: 2nd Nordic Symposium on Cloud Computing and Internet Technologies. pp. 38–45. ACM, Oslo, Norway (2013)
9. Forum, O.G.: Open Cloud Computing Interface (OCCI), `http://occi-wg.org/`
10. Goncalves, G., Endo, P., Santos, M., Sadok, D., Kelner, J., Melander, B., Mangs, J.E.: CloudML:An Integrated Language for Resource,Service and Request Description for D-Clouds. In: IEEE 3rd CloudCom. pp. 399–406. IEEE, Athens (Nov 2011)
11. 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. SciTePress, Noordwijkerhout (2011)
12. Hermans, F., Pinzger, M., van Deursen, A.: Domain-Specific Languages in Practice: A User Study on the Success Factors. In: Model Driven Engineering Languages and Systems, pp. 423–437. Springer Berlin Heidelberg, Berlin (2009)
13. Kolovos, D., Rose, L., Abid, S., Paige, R., Polack, F., Botterweck, G.: Taming emf and gmf using model transformation. In: Model Driven Engineering Languages and Systems, LNCS, vol. 6394, pp. 211–225. Springer Berlin Heidelberg (2010)
14. Liu, D., Zic, J.: Cloud#: A Specification Language for Modeling Cloud. In: IEEE 4th CLOUD. pp. 533–540. IEEE, Washington, DC (Jul 2011)
15. Loutas, N., Peristeras, V., Bouras, T., Kamateri, E., Zeginis, D., Tarabanis, K.: Towards a Reference Architecture for Semantically Interoperable Clouds. In: IEEE 2nd CloudCom. pp. 143–150. IEEE, Indianapolis (Nov 2010)
16. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys 37(4), 316–344 (Dec 2005)
17. Petcu, D.: Multi-Cloud: expectations and current approaches. In: MultiCloud '13. pp. 1–6. ACM Press, Prague (2013)
18. Petcu, D., Macariu, G., Panica, S., Crăciun, C.: Portable Cloud applications—From theory to practice. Future Generation Computer Systems (Jan 2012)
19. Ranabahu, A., Maximilien, E.M., Sheth, A.P., Thirunarayan, K.: A domain specific language for enterprise grade cloud-mobile hybrid applications. In: SPLASH '11 Workshops. pp. 77–84. ACM Press (Oct 2011)
20. Satzger, B., Hummer, W., Inzinger, C., Leitner, P., Dustdar, S.: Winds of Change: From Vendor Lock-In to the Meta Cloud. IEEE Internet Computing 17(1), 69–73 (Jan 2013)
21. Sheth, A., Ranabahu, A.: Semantic Modeling for Cloud Computing, Part 1. IEEE Internet Computing 14(3), 81–83 (May 2010)
22. Silva, G.C., Rose, L.M., Calinescu, R.: A Systematic Review of Cloud Lock-In Solutions. In: IEEE 5th CloudCom. pp. 363–368. IEEE, Bristol, UK (Dec 2013)