DECOUPLING CLOUD APPLICATIONS FROM THE SOURCE
A Framework for Developing Cloud Agnostic Software

Joaquín Guillén[1], Javier Miranda[2] and Juan Manuel Murillo[2]
[1]*GloIn, Calle Azorín 2, Cáceres, Spain*
[2]*Department of Information Technology and Telematic Systems Engineering, University of Extremadura, Cáceres, Spain*

Keywords: Cloud Computing, Utility Computing, Cloud Lock-in, Framework, Decoupled Cloud Applications.

Abstract: Cloud computing and the utility computing model have aroused the interest of multiple vendors to provide their own public cloud services. Each vendor provides different services and establishes a series of restrictions for all of the applications deployed within its infrastructure, which results in cloud applications being modeled and developed for specific cloud environments. This leads to a tight coupling of applications to the cloud in which they are deployed, thus complicating their migration to other clouds. In this paper a different approach for developing cloud applications is proposed, based on the separation of cloud related metadata from the source code that comprises an application deployed in a cloud. Separating metadata related to how services are provided by the application and how it consumes cloud specific and remote services will allow developers to be oblivious as to which cloud the application is being developed for. This approach may be used both for developing new cloud applications as well as for migrating legacy software to the cloud.

1 INTRODUCTION

Cloud computing has evolved considerably during the past years into an alternative for many companies as a means of hosting and maintaining their enterprise applications (Wang et al., 2010). Most public cloud providers follow a utility computing business model which allows cloud users to use *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) or *Software as a Service* (SaaS) services that are provided on a pay-as-you-go basis (Armbrust et al., 2009). The technical and financial models adopted by cloud computing have awakened the interest of many companies that have seen in cloud computing an alternative to maintaining private datacentres and a means of reducing costs. Clouds allow companies to externalise their system maintenance processes as well as providing them with a scalable solution regarding the computing resources they may require in any given moment and the costs of using these resources.

An increasing demand for cloud services has urged companies such as Amazon (Amazon, 2011), Google (Google, 2011), Microsoft (Microsoft, 2011) and IBM (IBM, 2011) to release their own public clouds. The existent clouds allow their users to deploy applications based on a series of services and requirements that are independently defined by each cloud provider (Maximilien et al., 2009). As a result of this, applications developed for the cloud are tightly coupled to the cloud infrastructure or platform for which they have been developed (Tsai et al., 2010).

Having to develop applications that are tightly coupled to a particular cloud presents a problem for companies. Lower service costs, better SLAs or even customer demands may lead a company to consider porting their cloud applications from one cloud to another. In most cases this will require redesigning and redeveloping the software in order to adjust it to the targeted cloud's requirements (Tsai et al., 2010). Considering that the heterogeneous cloud model provided by the industry is accompanied by the previously mentioned restrictions, we feel that additional tools are required for building cloud applications.

This paper proposes an approach for building applications that are loosely coupled to clouds. It proposes a framework that allows software to be developed without being conditioned by the inherent requirements and limitations of the cloud in which the application will be hosted. Information about

70 Guillén J., Miranda J. and Manuel Murillo J..
DECOUPLING CLOUD APPLICATIONS FROM THE SOURCE - A Framework for Developing Cloud Agnostic Software.
DOI: 10.5220/0003901700700075
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 70-75
ISBN: 978-989-8565-05-1
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

how applications access both the cloud's services as well as services provided by external applications is separated from the source code and managed by the framework. The adoption of this software development framework for cloud applications will not only be intended for building new applications; legacy software may also be migrated to the cloud without having to redevelop most of its components.

The paper is organized as follows: Section 2 identifies several concerns that motivate our proposal. Section 3 describes our proposal; in this section we present a model containing cloud specific metadata that must be managed separately from cloud applications' source code. Section 4 provides information about the most relevant work related with our proposal. Finally Section 5 gathers our conclusions and the limitations that our work has, this way presenting future work that will have to be done.

2 MOTIVATION

Cloud computing service providers for IaaS and PaaS type clouds declare a series of services and restrictions aimed towards software developed for their architecture. By providing developers with tools and APIs that are integrated into each cloud's specific development process, cloud vendors guarantee that software that is fully compliant with their cloud's requirements is built: Google App Engine (Google, 2011), Amazon Web Services (Amazon, 2011) and Microsoft Azure (Microsoft, 2011) are clear examples of this. They all provide tools and SDKs for developers to be capable of creating software for their clouds (Zhang, 2010). By using these tools applications will be provided with cloud specific libraries used for invoking cloud specific services. This approach presents two main inconveniences:

- Cloud specific service invocations will have been hard coded into the software.
- Direct dependencies will be created between the developed application's classes and those provided by the SDKs.

This creates a strong coupling between the software and the underlying architecture which will prevent applications from being migrated to different clouds. Consequently a problem arises related to how applications interact with their underlying architecture as well as with remote clouds (Knorr et al., 2008; Armbrust et al., 2009).

In most cases, cloud applications will have to interact with other applications that may be hosted in different clouds. In such cases the remote services will be accessed in a different manner, depending on the cloud in which they are hosted, i.e. resources may be made available via SOAP, REST, publish/subscribe messaging, etc. This type of restriction introduces several problems in the process of developing software for the cloud:

- Applications have to provide their services based on the technological restraints of the cloud environment in which they are hosted.
- Applications have to consciously publish their services based on the technological restraints of the cloud environment in which they are hosted. For example, if the cloud only allows SOAP services developers will have to code a SOAP service.
- Applications have to consciously consume services based on the technology in which they are provided. For example, if the service is provided as a SOAP service developers will have to code a SOAP client.
- Applications will only be capable of consuming services that comply with the set of service types supported by the cloud in which they reside.

Considering that companies may choose to migrate their software from one cloud to another based on customer demands, cost reduction purposes and SLAs (Buyya et al., 2008), these limitations must be treated cautiously. Under these circumstances porting an application to a new cloud will not only require redeveloping that application, it may also require redeveloping all of its dependant applications. This problem may be aggravated if the application has to be divided into components that are deployed in different cloud platforms. In this case developers will have to integrate each component into its correspondent cloud as well as to redesign each individual component defining it as a collection of service providers and consumers.

The motivations gathered in this section lead us to consider that developing software for the cloud must be complemented with a series of vendor independent tools. These tools must allow developers to decouple their software from specific clouds, hence relieving the lock-in problem that cloud users are exposed to (Chow et al., 2009).

3 APPROACH

As we mentioned previously, this paper proposes a different approach for developing software for cloud platforms aimed at decoupling the software from its underlying stack. A cloud development framework is presented as a means of attaching cloud agnostic applications to specific cloud platforms. The framework has been conceived based on a series of principles that are commonly used in software engineering for developing quality software. These principles are enumerated in the following points:

- **Inversion of control (IoC):** this architectural principle is an inherent feature of most software development frameworks. It provides frameworks with a means of controlling the behaviour of applications in order to deal with specific issues (Fowler, 2004).
- **Dependency injection (DI):** dependency injection is a specific form of IoC, in which objects use other objects in the system without being aware of how they were created. Used together with polymorphism it allows an object to be unaware of the exact behaviour of the object it is using (Fowler, 2004).
- **Obliviousness:** the obliviousness principle is a consequence of using the two previous principles; it allows developers to build applications without having to consider concerns that are taken care of by the framework. This principle was originally introduced in the scope of Aspect Oriented Programming (Filman and Friedman, 2000).
- **Service adapters:** service adapters introduce an intermediate layer between a service consumer and a service that may be used for many purposes such as data transformation.

The main idea behind our proposal is to separate all of the dependencies between the developed application and the cloud from the source code by using the principles that we have enumerated. To do so we have defined a data structure that will contain information related to how each application will interact with its underlying architecture as well as with external applications hosted by other clouds. We have chosen to represent this data structure as a XSD diagram because it offers a simple way of representing data that is not associated to a specific programming language.

In Figure 1 we appreciate how an application developed for the cloud will be defined as a combination of service providers (*cloud-provider* elements) and service consumers (*cloud-consumer* elements). The following subsections explain how the principles that we previously mentioned are combined with this data structure in order to decouple applications from specific cloud platforms.

3.1 Oblivious Service Providers

A *cloud-provider* element will allow the framework's inversion of control mechanism to publish a service without it being hard coded into the source code.

For each *cloud-provider* element a service will

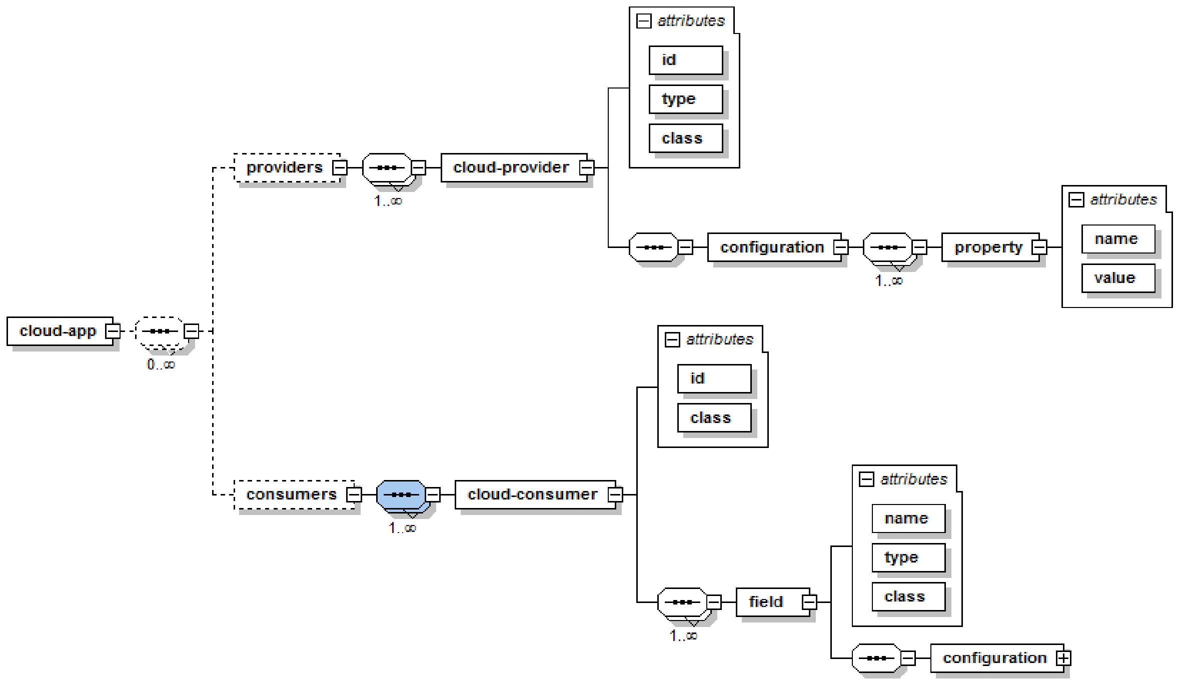

Figure 1: Data structure for cloud related metadata.

be published based on the class identified by the *class* attribute. The technology in which the service is provided, which must be supported by the cloud, will depend on the value of the *type* attribute. Publishing each different type of service will require a different set of configuration parameters. This is why a generic name-value configuration structure has been chosen to configure each service.

By following this approach services can be published without having to implement them in the application's source code. As a result applications are further decoupled from the cloud in which they are hosted.

3.2 Oblivious Service Consumers

Cloud applications will behave as service consumers for two types of services: remote services and local services. The latter may be provided by the cloud infrastructure or by other applications hosted by the same cloud. In order to avoid coupling the application to its dependencies the proposed framework allows developers to consume services by injecting dependencies into class fields.

A *cloud-consumer* element will be used as part of the cloud application's metadata to configure the services used by any class. Each *cloud-consumer* element will have an id attribute, which identifies the consumer in the framework, as well as a class attribute, which will be used to associate the consumer to a specific class of the application. Additionally a sequence of field elements will be declared as part of the cloud consumer which will define the service clients that will be injected into different class fields. Each field element will have a *name* attribute**,** containing the name of the field in which a service client will be injected, a *type* attribute**,** indicating the type of service that will be consumed by the client, i.e. SOAP, REST, CORBA, DDS, and a *class* attribute containing the class name of the instance that is being injected into the field.

3.3 Service Adapters

Service adapters are proposed as an additional means for decoupling the developed software from the underlying architecture as well as from other consumed services.

Considering that the services and APIs exposed by different cloud providers can be grouped into different categories, i.e. security, persistence, traceability, etc. We propose to provide adapters for these services that will behave as service clients injected into class fields. These adapters will be configured in the same way as we explained in the previous section. They will provide a common interface for connecting an application to the underlying cloud, but will however have specific implementations for each cloud.

This approach will also be used to adapt applications to local and remote services in those cases in which data transformations or service mappings have to be applied.

3.4 Legacy Software

Companies may choose to migrate an existent enterprise application to one or several clouds. The following steps will have to be taken to migrate legacy software to the cloud:

- **Identify cloud applications:** the architect will have to decide how the application will be migrated to the cloud. He may decide to use a single cloud or to migrate each tier or logical component to a different cloud.
- **Create independent projects:** the legacy software will have to be divided into separate software projects, in which each project will be hosted by a different cloud environment.
- **Identify services provided by the framework for each project:** some of the services consumed by the legacy software may already be provided by the framework (persistence, security, logging, etc.). In such cases *cloud-consumer* elements will have to be setup in order for the application to inject these dependencies into the legacy software. If the services offered by the framework cannot easily be integrated into the legacy software, a service adapter may be implemented.
- **Identify service providers for each project:** each new cloud application will have to provide a series of services. These services will have to be identified and included as part of the cloud application's metadata following the process explained in section 3.1.
- **Identify service consumers for each project:** each new cloud application will consume a series of services. The service consumers will have to be identified and included as part of the cloud application's metadata, following the process explained in section 3.2.

As a result of following these steps legacy software can be migrated to one or several different clouds. Following this approach will be a lot less intrusive in the resultant code, since no dependencies will be created towards a specific cloud.

4 RELATED WORK

The lock-in problem that cloud developers are exposed to when developing applications is a recognised problem (Chow et al., 2009) that the research community is trying to solve. The solutions provided are commonly oriented towards modeling applications for the cloud, or providing cloud architectures that will allow applications to be migrated between clouds. However, as far as we are concerned, little work has been done to decouple the final source code from the cloud's infrastructure.

The work done in (Maximilien et al., 2009) is probably the most related to our work. It proposes to use a middleware that behaves as a cloud broker for cloud clients. The middleware provides tools and REST based APIs for deploying and consuming the services exposed by each cloud; these are based on a meta-model defined in the work as an abstract representation of cloud functionalities. Applications use the mechanisms provided by the middleware APIs to invoke services that may be hosted in different clouds. The goals of their work are similar to ours, however the approach is different, considering that intermediate management software is placed between the applications and the clouds. Additionally a tight dependency is created between the applications and the middleware.

An approach for modeling cloud applications is proposed by (Hamdaqa et al., 2011). In this work a meta-model for cloud applications is defined which is centralized in the definition of a cloud task as *a* "composable unit, which consists of a set of actions that utilize services to provide a specific functionality". The proposed meta-model is intended to decouple the design process from specific cloud platforms, however the approach does not cover decoupling the source code from cloud platforms. The proposal presented by (Frey et al., 2010) also uses models; in this case the *CloudMIG* approach is presented as a means of mapping models of existing cloud environments to legacy software models and transforming the result to cloud-specific code through a series of iterations and result evaluations. The source code generated by this proposal will be tightly coupled to the cloud platform which it was aimed at.

Another proposal that aims to allow clouds to interact with each other is found in (Tsai et al., 2010). In this case a Service Oriented Cloud Architecture is proposed. The architecture introduces an ontology mapping layer over the services published by each individual cloud platform as a means of *masking* the differences between each individual cloud provider. Cloud brokers deploy applications in one cloud or another depending on the budget, SLAs and QoS requirements that are negotiated with each provider. No restrictions are made on how applications are developed; they may still be tightly coupled to specific clouds and may be developed with APIs provided by cloud providers.

All of these works allow developers to work around the lock-in problem; they provide a series of tools that make more flexible the process of developing software for the cloud. However in all of them developers continue to generate tightly coupled software that cannot easily be migrated from one platform to another or from cloud to non-cloud environments.

5 CONCLUSION AND FUTURE WORK

In this paper a cloud development framework has been proposed that will allow developers to create applications that are not tightly coupled to a specific cloud. Cloud-related data about how the applications use the SDKs and APIs provided by each vendor and about how each application behaves as a service provider and consumer, is separated from the source code. As a result of this each application's source code is decoupled from its underlying architecture.

One of the main problems that companies have to confront when it comes to deciding whether they want to shift their business processes towards the cloud is deciding whether the lock-in problem makes it worth the while. Work is being carried out among the research community to come up with solutions for this problem; however the solutions provided often propose changing the architectures used by cloud providers or changing the software development processes used by companies. These types of solutions are not easily adopted by cloud providers and users; in the first case because providers are not interested in standardization, and in the second case because users are not willing to change their software development processes.

We believe that tools have to be provided in order to make the cloud technology much more accessible and flexible for providers and developers. These tools should contribute to clarify the doubts that many companies have regarding the cloud, and allow them to rely on cloud computing as a viable means of hosting their applications, thus allowing cloud computing to exploit its full potential. This was the idea that inspired us into writing this paper: proposing a way of creating decoupled software for the cloud without introducing any foreign elements into the existent technology.

Future work involves extending the proposed framework in order to manage QoS settings for each cloud as well as enhancing the prototype of the framework. Additionally we would like to complement the framework with tools that will allow developers to choose which cloud the software will be deployed in and automatically generate a cloud specific configuration, without having to manually set the provider and consumer elements. To do so we plan to use Model-Driven Software Development to define a cloud meta-model that will allow designers to model an application that will be hosted in the cloud. The source code generated from these models will contain a preconfigured cloud file containing the application's metadata. For those cases in which applications are not created from models, we plan to build a plugin for the Eclipse IDE which will assist developers in the process of configuring the application's cloud-related metadata.

ACKNOWLEDGEMENTS

Spanish Ministry of Science and Innovation under Project TIN2011-24278.

Junta de Extremadura, Fondos FEDER.

REFERENCES

Amazon (2011). Amazon Web Services, [Online], Available: http://aws.amazon.com [14 Nov 2011].

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep.* UCB/EECS-2009-28.

Buyya, R., Yeo, C. S., Venugopal, S. (2008). Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. *2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5-13.

Chow, R., Golle, P., Jakobsson, M., Shi, Staddon, J., Masuoka, R., Molina, J. (2009). Controlling data in the cloud: outsourcing computation without outsourcing control. *In Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW*, pages 85-90.

Filman, R. E. & Friedman, D. P., 2000. Aspect-Oriented Programming is Quantification and Obliviousness. October, 2000(May), p.1-9.

Fowler, M., 2004. Inversion of Control Containers and the Dependency Injection pattern. I Can, M(1), p.1-19. Available: http://www.martinfowler.com/articles/injection.html.

Frey, S., Hasselbring, W. (2010). Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach. In Cloud Computing 2010: *Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 155-158.

Google (2011). Google App Engine, [Online], Available: http://code.google.com/appengine/ [14 Nov 2011].

Hamdaqa, M., Livogiannis, T., Tahvildari, L. (2011). A Reference Model for Developing Cloud Applications. *In Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pages 98-103. SciTePress.

IBM (2011). IBM Smart Cloud Foundation, [Online], Available: http:// www.ibm.com/cloud-computing/ [14 Nov 2011].

Knorr, E., Grumman, G. (2008). What cloud computing really means [Online]. Available: http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031.

Maximilien, E. M., Ranabahu, A., Engehausen, R., Anderson, L. C. (2009). Toward cloud-agnostic middlewares. In OOPSLA09: *14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 619–626.

Microsoft (2011). Microsoft's Windows Azure, [Online], Available: http://www.microsoft.com/windowsazure/ [14 Nov 2011].

Tsai, W., Sun, X., Balasooriya, J. (2010). Service-Oriented Cloud Computing Architecture. In ITNG10 *7th International Conference on Information Technology: New Generations*, pages 684-689.

Wang, L., Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., and Fu, C. (2010). Cloud Computing: a Perspective Study. *New Generation Computing*, 28(2), pages 137-146.

Zhang, Q., Cheng, L., Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1), pages 7-18.