A REFERENCE MODEL FOR DEVELOPING CLOUD APPLICATIONS

Mohammad Hamdaqa, Tassos Livogiannis and Ladan Tahvildari
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada

Keywords: Cloud computing, Reference model, Meta-model, Software architecture, Model-driven architecture.

Abstract: Cloud Computing is a paradigm shift that involves dynamic provisioning of shared computing resources on demand. It is a pay-as-you-go model that offers computing resources as a service in an attempt to reduce IT capital and operating expenditures. The problem is that current software architectures lack elements such as those related to address elasticity, virtualization and billing. These elements are needed in the design of cloud applications. Moreover, there is no generic cloud software architecture for designing and building cloud applications. To further complicate the problem, each platform provider has different standards that influence the way applications are written. This ties cloud users to a particular provider. This paper will focus on defining a reference model for cloud computing; more particularly, it presents a meta-model that shows the main cloud vocabulary and design elements, the set of configuration rules, and the semantic interpretation. It is always important to understand the abstract architecture of a system, and then tackle platform-specific issues. This separation of concerns allows for better maintainability, and facilitate applications portability.

1 INTRODUCTION

Cloud computing is paving its way into the enterprise. In its early years, the cloud computing research was focused on building robust cloud infrastructure. Mainly, it focused on improving cloud datacenters and its related technologies that allow for scalability and elasticity. After building the underlying cloud infrastructure, the hype was building platforms on top of the cloud infrastructure. Today, a number of companies (e.g., Microsoft, Google, and Amazon) provide platforms that allow for efficient development of cloud applications and easy control of data. Offering a pay-as-you-go billing policy ties the operating expenditure to the provider's offer. Selecting the best offer may dictate a shift from one provider to another; the need to partially redevelop the application makes this shift difficult and more costly. The main concerns of cloud providers are how to deal with these issues, in particular standardization and interoperability between different cloud platforms (Tsai et al., 2010). Vendor Lock-in, or being tied to a specific vendor deployment environment is what hinders the decision of many customers to move to the cloud. This challenges cost reduction and portability across multiple vendors. This paper addresses the above issues by providing a reference model for developing cloud applications. A good design requires having the right elements and vocabulary, which match the implementation elements. Our preliminary study of two of the main cloud platform development environments, namely Windows Azure (Microsoft, 2010) and Google App Engine (GAE) (Google, 2010), uncovered the fact that both platforms share some common components, despite the fact that they are built based on different business and pricing models.

The work this paper presents is part of a larger project. Its aim is to define a cloud software architecture, a cloud modeling language, and its related design patterns. The goal is to allow cloud users (Armbrust et al., 2009) to design applications independent of any platform and to build inexpensive elastic applications.

From a software engineering point of view, a cloud application is a software provided as a service. It consists of the following: a package of interrelated services, that we later will call *tasks*, the definition of these tasks, and the configuration files, which contain dynamic information about tasks at run time. Cloud tasks provide compute, storage, communication and management capabilities. Tasks can be cloned into multiple virtual machines, and are accessible through application programmable interfaces. Cloud applications are a kind of utility computing that can scale out

Hamdaqa M., Livogiannis T. and Tahvildari L..
A REFERENCE MODEL FOR DEVELOPING CLOUD APPLICATIONS.
DOI: 10.5220/0003393800980103
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 98-103
ISBN: 978-989-8425-52-2

and in to match the workload demand. Cloud applications have a pricing model that is based on different compute and storage usage, and tenancy metrics.

In the next section we present related work. Section 3 justifies the need of a cloud application reference model. We define and explain the cloud application reference model in Section 4, followed by a conclusion and future directions in Section 5.

2 RELATED WORK

Perhaps the work done in (Tsai et al., 2010) is the most related to our work. Tsai et al. proposed a "Service-Oriented Cloud Computing Architecture" that allows cloud applications to work with each other. The proposed architecture is a three layered architecture that consists of a *cloud ontology mapping layer*, a *cloud broker layer*, and a *SOA layer*. The work we are presenting in this paper falls under the ontology mapping layer where we define a reference model, providing the main vocabulary of cloud applications and the relations between them. The reference model takes into consideration the multi-tenancy pattern (the *Single Application Instance and Multiple Service Instances*) presented by the authors.

A model-driven approach for building cloud solutions is also presented in (Charlton, 2009). Three design goals, which are similar to our goals, are presented. These goals are the following: *the separation of applications from infrastructure*, *the enablement of computer-assisted modeling and control automation*, and *the explicit collaboration to enact changes*. In the same publication, the authors denoted eight characteristics that a cloud application should incorporate, in order to achieve the above-mentioned goals. Furthermore, this industrial paper introduced and briefly described three Elastic Modeling Languages (EML) for computing, deployment and management of elastic applications. The reference model we are presenting in this paper acts as a meta-model for such languages and conforms with the goals presented in this paper.

In (Frey and Hasselbring, 2010), CloudMIG, a framework to facilitate the migration of legacy software systems to the cloud is presented. Six actions are proposed for this migration to address four shortcomings that the authors state as problems in the migration process. Both our approach and CloudMIG are based on model driven engineering. While our approach starts from current cloud platforms to extract common vocabulary and elements to create a cloud meta-model, CloudMIG starts from existing legacy systems, extracts the actual architecture, and then uses the selected target cloud platform meta-model along with a utilization model to generate a target model toward system migration.

Variation analysis is the technique proposed in (Zhang and Zhang, 2009) as a way of designing cloud applications. The authors defined a set of Architectural Building Blocks and ways to assemble a cloud-SOA solution from them, using variation analysis. This approach is specifically designed and checked for Service Oriented Architecture software systems built using the Service-Oriented Solution Stack, a template layered architecture from IBM.

The need to detach the cloud application development process from specific cloud platforms is addressed in (Maximilien et al., 2009). A platform-agnostic middleware is proposed. This middleware lies on top of the Platform-as-a-Service layer. It provides API and services to be used by cloud users, and transparently deploys the application to a specific (but initially unknown) cloud platform. Interestingly, this approach relieves developers from the cloud vendor lock-in problem, but ties the development of cloud applications to the proposed middleware.

A bottom-up approach for assembling cloud applications from simpler components, the MacroComponents is presented in (Matthews et al., 2009). The proposed approach leverages a number of open source technologies, in order to provide a component model for building cloud applications. Open source technologies involved, include the OSGi component model, the P2 provisioning infrastructure for OSGi component model, and the CloudClipse, which is an eclipse plugin for managing the deployment and installation of specific virtualization images used in cloud platforms, such as Amazon EC2 or Eucalyptus.

From an IT management perspective, (CA Labs, 2009) proposed a cloud architecture to facilitate compatibility between ITIL and cloud computing, as well as portability of cloud applications between different cloud vendors. The goal is to maximize the return of IT investment (ROI) in cloud computing. The approach's foundation is the C3A paradigm. The reference architecture consists of specific components, which enable provisioning of application's agreement on SLAs, and the migration of the application to different cloud vendors. The proposed architecture is essential for migrating ITIL compatible applications to the cloud, and managing existing cloud applications. Nevertheless, our paper presents a reference model rather than a reference architecture. The proposed reference model facilitates the cloud application development, from the design to the implementation, in a transparent way, without depending on specific Platform-as-a-Service or Infrastructure-as-a-

Service components.

The problem of migrating legacy software systems is the main topic of (Zhang et al., 2009). The authors provide a seven step methodology, inspired from the SEI's horseshoe model, to migrate legacy software systems to cloud platforms. The methodology involves re-engineering the original legacy architecture, refining it to a modern service oriented architecture, and porting it to the cloud using MDA transformation techniques. Our cloud reference model can add value to this methodology, by providing a unique targeted model for the transformations, before the deployment of the re-factored legacy application to the cloud. This can improve the approach's flexibility, especially that the original methodology forces developers to choose a cloud vendor before porting the legacy application to the cloud.

3 THE NEED OF A CLOUD APPLICATION REFERENCE MODEL

Before listing the cloud application reference model components and their relations, it is important to justify the need for this model and the difference between this model and SOA reference model.

SOA is an umbrella that describes any kind of service. A cloud application is a service. A cloud application meta-model is a SOA model that conforms to the SOA meta-model. This makes cloud applications SOA applications. However, SOA applications are not necessary cloud applications. A cloud application is a SOA application that runs under a specific environment, which is the cloud computing environment (platform). This environment is characterized by horizontal scalability, rapid provisioning, ease of access, and flexible prices. While SOA is a business model that addresses the business process management, cloud architecture addresses many technical details that are environment specific, which makes it more a technical model.

Cloud platforms are complex environments, which need to be refined at different levels of granularity. The cloud hierarchical view (i.e. SaaS, PaaS, IaaS) is an example of a refinement that uses SOA to describe the high level services provided over the internet (the cloud). There is a need to create a modeling language that is tailored to build efficient, elastic and autonomous applications from tasks and services provided by the cloud environment, and to define patterns that can result in the efficient optimization of money and resources.

The modeling language should be platform independent, with enough technical details that allow it to tackle the platform specific environments. This will facilitate the task of the design and implementation of application in these environments. This will also help in migrating applications between different cloud providers, or deploy the same implementation on different platforms. This programming model should also provide programmers with the best practices that can help in the design and implementation of cloud applications.

Any model consists of a vocabulary of design elements, a set of configuration rules, and a semantic interpretation. In the following section, we will define the cloud computing architecture by drawing a meta-model that represents the main components of cloud applications and the relations between them.

4 THE CLOUD APPLICATION META-MODEL

Figure 1 is a meta-model of cloud applications. A *Cloud Application* is neither pure service oriented nor a standard web application. Cloud applications combine traditional software with remote services to provide highly available, scalable and high performance software, in addition to a fault tolerance, scalable storage that can scale to petabytes. A cloud application supports the Software plus Services (S+S) paradigm (Sirtl, 2008), where each application consists of a number of *Tasks*.

A *Cloud Task* is a composable unit, which consists of a set of actions that utilize services to provide a specific functionality to solve a problem. It is a mutated unit that can be copied to other virtual machines in order to allow horizontal and vertical scalability. Tasks should satisfy when composed the following principals: statelessness, low coupling, modularity, and semantic interoperability. *Task*s are semantically connected to other tasks in the cloud through the *roles* they play in order to satisfy a specific business requirement, which is bounded by obligations or *responsibilities*. Cloud tasks are uniquely identified by a global Dynamic Name Service (DDNS) that can be assigned to a dynamic virtual IP address at run time. This makes the task highly available and fault tolerant, and allows the cloud application to be dynamically upgradable without any interrupts.

Like any other service or component, a *CloudTask* has a definition file. A *TaskDefinition* stores information about tasks that the cloud application provides. A *TaskDefinition* contains information about the cloud application tasks, which are determined at

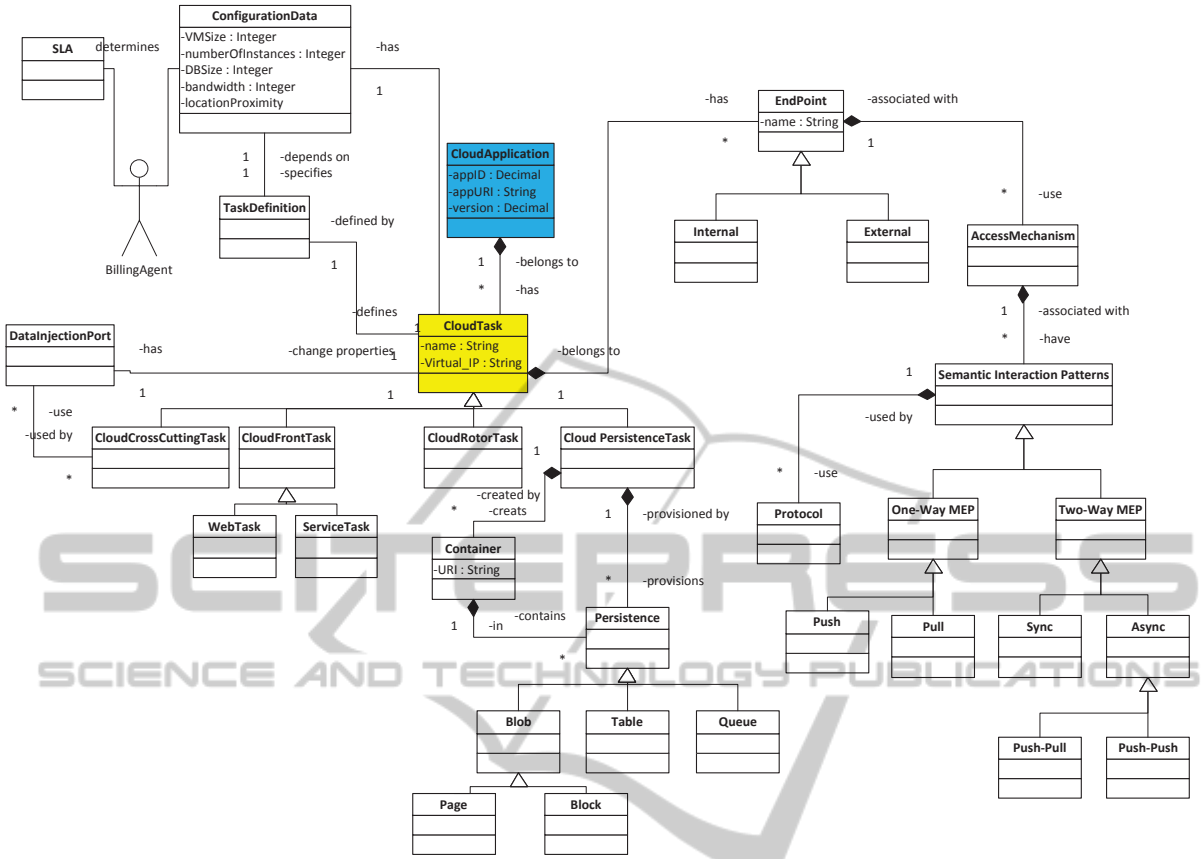

Figure 1: Cloud Application Meta-Model.

design time. A *TaskDefinition* provides the structure of the cloud application, in terms of the provided tasks, their types and relationships. It also provides the set of task interfaces and their contracts.

What makes a cloud application different from other applications is its elasticity. Cloud applications have the ability to scale out and in. This can be achieved by cloning tasks into multiple virtual machines at runtime to meet the changing work demand. *ConfigurationData* is where dynamic aspects of cloud application are determined at runtime. There is no need to stop the running application or redeploy it in order to modify or change the information in this file. *ConfigurationData* contains information such as the size of the virtual machine (*VMSize*), number of instances (*numberOfInstances*), database size (*DBSize*), *bandwidth*, even the location (*locationProximity*) where you want your task instances to run and whether they belong to the same affinity group or not. Different cloud platform providers have different pricing models. The general umbrella is pay-as-you-go. However, the way resources are allocated based on the amount of money you pay is different. Some cloud platform providers allow the cloud user (application developer) to allocate resources explicitly. The cloud user can set the values in the *ConfigurationData* file based on need and budget. Other cloud platform providers provide algorithms *(BillingAgent)* to dynamically allocate resources based on the amount of money a cloud user is willing to pay. The user can set the budget and general guidelines, and the provider will modify the values in the *ConfigurationData* file automatically for the best configuration. The cloud provider uses the *ConfigurationData* as a contract with the cloud user. This can be represented in a clear readable format as a Service Level Agreement (*SLA*).

Task properties can also be modified at runtime. This can be achieved through a *DataInjectionPort*. A *DataInjectionPort* modifies tasks crosscutting properties such as those related to quality of service (QoS). Cloud platform providers vary in the way they support data-injection. This is because data-injection techniques are considered risky and can be a source of many security threats.

CloudTask can be classified into:

a ***CloudFrontTask.*** An entry point to the cloud application that handles user requests, which are dis-

tributed by a load balancer. A *CloudFrontTask* must support the interactive request-response pattern. It is usually a web application (*WebTask*) hosted on the cloud datacenter where a web-server is always enabled. However, it can also be a web-service (*ServiceTask*), which is provided by a third party. A *ServiceTask* uses the Enterprise Service Bus (EBS) to discover and access remote or enterprise services.

b ***CloudRotorTask*.** This task runs in the background of the *CloudFrontTask* on the cloud datacenter. It is not directly accessible from outside the cloud datacenter. Mainly, it does some general development work, or helps other tasks by performing a particular functionality. *CloudRotorTask* must support event-driven communication patterns. Grid computing tasks are common examples for *CloudRotorTask*s.

c ***CloudCrossCuttingTask*.** This task is responsible for managing cross-cutting aspects such as those related to monitoring cloud resources, which includes compute and storage instances and a load balancer to ensure resource utilization and performance. It is also responsible for logging, maintaining quality of services of the cloud application, deployments of application/tasks, dynamically add/remove instances based on metrics, launching instances, log-in to instances, and task properties change through the *DataInjectionPort*. *CloudCrossCuttingTask*s can be accessed directly through a web portal or a specific API (i.e. REST, SOAP). Communication with *CloudCrossCuttingTask*s should be secure by applying, for example, one of the public key algorithms and by using certificates (i.e. HTTPS EndPoint)

d ***CloudPersistenceTask*.** The main role of *CloudPersistenceTask*s is to manage storage accounts. *CloudPersistenceTask*s manage the access control and login to cloud storages. A cloud storage (e.g., blob, table, queue) does not have any access control mechanism; it is the responsibility of the persistence task to provide the authorization and authentication services. *CloudPersistenceTask*s create containers, which are analogous to folders but with no nesting. Containers are accessible through a unique Uniform Resource Identifier (URI). *CloudPersistenceTask*s assign persistency to containers and give them a unique URI that is either privately or publicly accessible. The *CloudPersistenceTask* supports three main types of cloud storages that are reliable, can scale out, simple, inexpensive and have better performance under the cloud environment. These types are: unstructured data (blobs), structured data (tables) and asynchronous messaging (queues).

i **Blob.** Blobs are unstructured large data files and their meta-data. It can be stored as a sequence of blocks or pages. The blob is the simplest and largest cloud storage unit. Cloud drive storages are blobs.

ii **Table.** Tables are structured data files, that are more complex than blobs, but different than relational database (RDB) tables. Cloud tables are much simpler. This makes them suitable for massive scalability. They can scale out to support any number of simultaneous tasks. A cloud table is a set of entities and its associated properties. It uses two types of keys: partition keys and row keys. Cloud tables do not support SQL queries, have no schema and use optimistic concurrency for updates and deletions. Cloud tables are more like datasheet tables.

iii **Queue.** A scalable messages storage, which supports the polling-based model used in message passing between tasks. A message can be stored for long periods (i.e. days) before it is read and then removed from the queue. Cloud queues are different from conventional queueing systems. Cloud queues must support fault tolerance. Unlike conventional queues, a message read does not delete the message from the queue. The message is set into a hidden mode until it is successfully processed. It is the responsibility of the processing task to delete the message. The queue is the main communication mechanism between *CloudFrontTask*s and *CloudRotorTask*s. Which makes it one of the most frequently used design patterns in the cloud. This design pattern does not only relief the end-user from waiting for a long time until a task processes the message, but also makes scalability easier.

Relationships between tasks can be determined by *EndPoint*s. *EndPoint*s are ports through which a *CloudTask* can connect to other tasks or to the environment. Each Task has one or more *EndPoint*s. An *EndPoint* can be classified based on several criteria. Whether it is publicly visible (external) or only accessible within the Cloud Application (internal), load balanced at the network level or not, or whether it allows inbound or outbound communication. Each *EndPoint* uses an access mechanism, which uses a semantic interaction pattern for the coordination of message exchange. These patterns are based on specific protocols that determine the syntax and semantics of the messages that are exchanged between the

two communication parties. Message Exchange Patterns (MEP) can be classified into two main categories, one-way or two-way. The one-way MEP is usually referred to as the event driven MEP, or publish subscribe (pub/sub), in which the participating parties are not fully aware of each other. A temporary storage in the form of a queue is usually used to accomplish this. One party will push a message, and the second will pull it from the queue. This is one of the common communication patterns between *CloudFrontTask*s and *CloudRotorTask*s. On the other hand, the two way MEP is usually referred to as request/response MEP. It can be either a synchronous (blocking) or asynchronous (non-blocking). This is an interactive communication that is usually needed when you have direct interaction with the user. *CloudFrontTask*s must support this type of interaction with the application user.

5 CONCLUSIONS

This paper presented a meta-model for cloud applications. Cloud computing is a new paradigm for developing elastic and flexible applications with less time to market. The promise is to reduce the overhead of developing, configuring, deploying, and maintaining cloud applications. Currently, there is no common vocabulary, development methodologies, or best practices that distinguish the cloud development paradigm from the existing ones. The lack of standardization and common terminologies challenges portability and migration between different cloud platforms. On the other hand, the lack of software architectural models and design patterns makes cloud application development an ad-hock approach.

To address the previous problems, in this paper we defined a cloud application meta-model that is capable of capturing the syntax and some of the semantics of cloud applications. This meta-model can be used by developers to better understand cloud applications independent of any specific cloud development environment. This meta-model will serve as a first step toward a cloud modeling language that we are currently working on.

Future directions include refining the syntax and defining semantics of the proposed reference model, mapping the reference model to different cloud platforms, and creating a platform independent modeling-language for cloud applications.

REFERENCES

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*.

CA Labs (2009). Cloud computing Web-Services offering and IT management aspects. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 27–39.

Charlton, S. (2009). Model driven design and operations for the cloud. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 17–26.

Frey, S. and Hasselbring, W. (2010). Model-Based migration of legacy software systems into the cloud: The CloudMIG approach. In *WSR2010, 12th Workshop Software-Reengineering*, pages 1–2.

Google (2010). Google app engine. Retrieved: December 2010, from http://code.google.com/appengine/.

Matthews, C., Neville, S., Coady, Y., McAffer, J., and Bull, I. (2009). Overcast: Eclipsing high profile open source cloud initiatives. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 7–15.

Maximilien, E. M., Ranabahu, A., Engehausen, R., and 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 (2010). Windows azure microsoft's cloud service platform. Retrieved: December 2010, from http://www.microsoft.com/windowsazure/.

Sirtl, H. (2008). Software plus Services: New IT-and Business Opportunities by Uniting SaaS, SOA and Web 2.0. In *IEEE EDOC'08, 12th International Enterprise Distributed Object Computing Conference*, pages 1541–7719.

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

Zhang, L. J. and Zhang, J. (2009). Architecture-Driven variation analysis for designing cloud applications. In *IEEE CLOUD09, 2nd International Conference on Cloud Computing*, pages 125–134.

Zhang, W., Berre, A. J., Roman, D., and Huru, H. A. (2009). Migrating legacy applications to the service cloud. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 59–68.