TOWARDS A NEW PaaS ARCHITECTURE GENERATION

Claudio Guidi[1], Paolo Anedda[2] and Tullio Vardanega[1]
[1]*Department of Mathematics, University of Padova, Padua, Italy*
[2]*CRS4, Technology Park, Pula (CA), Italy*

Keywords: PaaS, IaaS, SaaS, SOA, Architectures, Services.

Abstract: In this paper we present our vision for a next-generation Cloud Computing PaaS layer intended for openness and federation. We base our vision on the principles of the Service Oriented Architecture paradigm, and discuss some design details of the internals of the PaaS we are currently prototyping.

1 INTRODUCTION

The PaaS is a fundamental layer of the Cloud Computing stack. It is the connection layer between the IaaS, the computing, storage and networking infrastructure, and the SaaS, where applications become available to the user. The PaaS provides the abstractions needed for developing applications abstracting away from the negotiation, contention and reclamation of the bare execution resources on which applications will eventually run. More specifically, the PaaS provides all the functionalities required for the development, the deployment and the monitoring of services at the level of SaaS. In our view, the PaaS also governs the resource requests that must be made to the underlying IaaS to meet the service level agreements separately defined with the SaaS and the IaaS: one between the user and the application provider; the other between the application provider and the infrastructure owner.

At present there is no clear picture of where the future evolution of Cloud Computing PaaS layer will lead. Current evidence shows that they will continue to be provided by big IT suppliers (e.g.: Google, Amazon, SalesForce). However, the scenario could as well significantly change if open-cloud ideas such as those proposed in (opencloudmanifesto.org, 2009) will meet with support. In the latter case the Cloud should be an open horizontal platform where different players can freely interact with one another, with obvious benefits for companies, developers and end users alike.

We contend that the role played by PaaS will be instrumental to determining how Cloud Computing will evolve in the future. One extreme is the dominance of closed solutions, such as those currently proposed by worldwide leaders (cf. e.g. Google App Engine (Google, 2011)), in which case Cloud Computing will be a huge vertical infrastructure controlled only by few actors. At the other extreme there is instead prevalence of open and standardized architectures, which enable an open Cloud scenario.

On account of these observations, it is reasonable to anticipate Cloud Computing PaaS layers realized as the mix of offerings from private and custom infrastructures and open solutions. As far as open infrastructures are concerned, interesting though initial proposals have been formulated by EU projects like IRMOS (EU, 2011a) and RESERVOIR (EU, 2011b), which prototyped demonstrative cloud infrastructures for the development of multimedia applications, and the deployment of services over the boundaries of different domains, respectively. Those results form a good starting point for furthering the concept of PaaS we envision. Yet we must acknowledge that, to the best of our knowledge, there does not currently exist any consolidated understanding of the PaaS framework architecture and rationale. There also is no standard definition of the interaction protocols between the IaaS and the PaaS, and between the PaaS and the SaaS.

The concept we present in this paper aims at proposing an initial model, termed PaaSSOA, for facilitating the development of next-generation PaaS frameworks. Our model captures the main functions which characterize the Cloud from the PaaS perspective and the interactions the PaaS has to have with the two adjacent levels in the Cloud SPI stack. Our proposal should be considered in the context of an open Cloud Computing infrastructure where the PaaS layer

Guidi C., Anedda P. and Vardanega T..
TOWARDS A NEW PaaS ARCHITECTURE GENERATION.
DOI: 10.5220/0003914102790282
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 279-282
ISBN: 978-989-8565-05-1

should be distributed, open and freely composed. The vision which inspires our effort focuses on the Service Oriented Architecture paradigm, which facilitates the construction of system architectures that are modular, scalable and standardized. The use of a Service Oriented Architecture for designing a PaaS reference model has two main advantages: (i) it allows for a standard and well-defined separation between the PaaS and the IaaS on the one hand, and between the PaaS and the SaaS on the other hand; (ii) it also warrants a high degree of flexibility, adaptability and modularity within the PaaS itself, which are very desirable architectural qualities. Interestingly, IRMOS (EU, 2011a) based its SPI vision on clear principles of service orientation, even though without providing a general rationale for realizing a SOA-like PaaS layer.

Besides providing a general reference for our PaaS concept, with the definition and prototyping of PaaSSOA we also investigate the mechanisms involved in the management of application-level services deployed on the IaaS in the form of a SOA: we refer to them as *meta-SOA mechanisms*. In Section 2 we present PaaSSOA by illustrating its architecture and main components. In Section 3 we discuss the issues related to the IaaS layer as seen from the perspective of PaaSSOA. In Section 4 finally we present some initial conclusions and plan for future work in this direction.

2 OUR VISION: PaaSSOA

Figure 1 depicts the main architecture of PaasSOA. In it, we focus on the relationship in place between the PaaS and the SaaS on the one hand, and between the PaaS and the IaaS on the other hand.

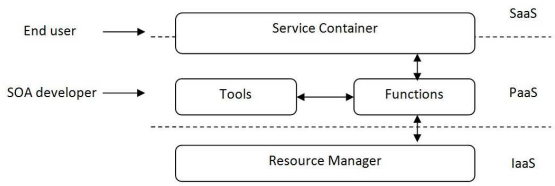

Figure 1: Main PaaSSOA architecture.

We see the PaaS layer as composed of three main blocks: *Tools* and *Functions*, which should be typical of a PaaS; and the *Service Container*, which is more specific of PaaSSOA. The Tools block provides the tools for designing and developing SOA and includes a web GUI designer for developers. The Functions block offers all the services which provide the more classic PaaS functionalities such as deployment and scheduling. The Service Container hosts the deployed services so that they can be reachable by both the end user and the developer. The end user needs to access the deployed services for ”consuming” them; the developer needs to monitor the deployed services to keep their performances under control. This is the reason why we have placed this particular block in the middle between the PaaS and the SaaS layers. Next we present and discuss the fundamental blocks of PaaSSOA.

Figure 2 depicts the architecture of the Tool and Functions blocks.

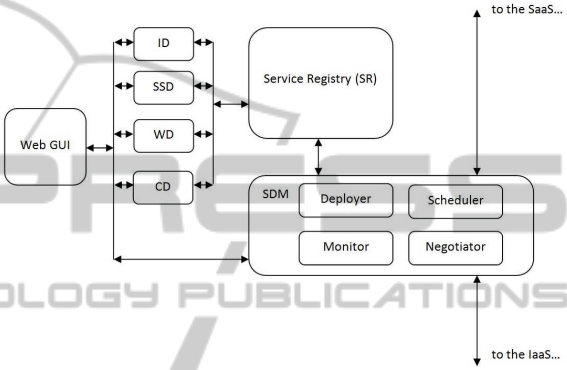

Figure 2: Tools and Functions architecture.

Blocks ID, SSD, WD, ConnD and CD represent the design tools needed for developing a SOA-based application or service. The function block is represented by the Service Registry (SR) block and the Service Deployment and Monitor (SDM) blocks.

The SOA tools we envision to supply with PaaSSOA include the following: *Interface Designer (ID)*, a graphical tool for designing service interfaces such as for example WSDL interfaces; *Simple Service Designer (SSD)*, a tool for developing simple services able to connect to data storage; *Workflow Designer (WD)*, a tool for designing orchestrators able to interact with other services; *Service Deployer and Monitor (SDM)*, an administrative tool for deploying services and monitoring them during execution; *Choreography Designer (CD)*: a tool for designing service choreographies, which address SOA system design from a global perspective, like in WS-CDL (Kavantzas et al., 2005).

As far as the function block is concerned, the SDM service implements some important tasks such as: *deployment*, to deploy new services or migrate the existing ones in different service containers; *scheduler*,to schedule service deployment depending on the available resources; *negotiator*, to negotiate resource allocations with the IaaS in accord with the stipulated the Service Level Agreement; *monitor*, to monitor the SLA conformance of the deployed services. In

case of under-performance, the Monitor can activate the scheduler and the deployer to reclaim the unused slack and assign it to respond to the request for new resources.

The Service Container block is part of the PaaS but it is also reachable also from the SaaS level as an endpoint for accessing deployed services. Moreover, the Service Container must be deployed within the virtual machine images allocated and controlled by the IaaS. The Service Container, termed SOABoot, starts at the boot of every individual virtual machine deployed for PaaSSOA. The SOABoot is a meta-service which manages a service registry for storing all the service descriptions (e.g. endpoint locations and interfaces) of the services executed in the corresponding virtual machine. The SOABoot is able to receive service implementations and store them as it is able to remove them. Moreover, it offers the possibility to remotely control the activation and the deactivation of the stored services. As shown in Figure 3, the set of all the SOABoot services represents the Service Container. The Deployer block manages all the SOABoots which form the Service Container.

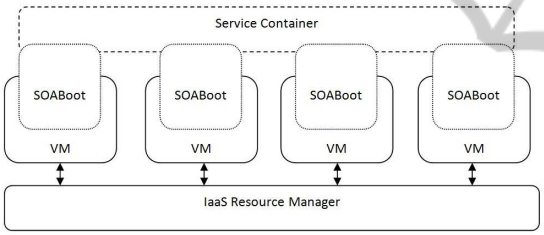

Figure 3: SOABoot architecture.

The key aspect of PaaSSOA is the high level of flexibility it aims to offer, as evidenced by the fact that it provides additional means for governing the efficiency and performance of service deployment from the SaaS, over those traditionally offered by the IaaS. In the event of under-performance detected in the deployed services, the PaaSSOA scheduler can take one of two actions: 1) request a new virtual machine (equipped with a SOABoot) to the IaaS so as to increase the set of bare resources availed to the user; 2) organize and manage the deployed services so as to optimize their individual and collective performance. for example it could move a service with low performance to another SOABoot which can guarantee a higher level of performance. Interestingly, this action can be taken without involving the IaaS.

Decision 2 makes it possible to implement optimization rules from within the PaaS level, without incurring dependence on the IaaS.

3 INTERACTING WITH THE IaaS LAYER

In a typical Cloud infrastructure, whether public or private, the system exploits the possibility to share the physical resources among the different virtual machines in order to minimize the waste of the unused resources and maximize the possibility of supporting multiple users. The many benefits introduced by multiplexing physical resources among different virtual machines cannot be doubted. However, without a precise policy for the governance of physical resource sharing, the performance of the virtual machines can be negatively affected by wrong utilization. For example, intensive access to disk and frequent request for exclusive access to physical resources (e.g., GPU, Infiniband card, etc.) can cause important delays and incur massive performance decay.

In order to build a flexible system and to support multiple technologies and actors, we want to maintain separation of concerns between the PaaS and the IaaS layers. Our approach consists in providing a bidirectional interface through which the two layers are able to constantly negotiate SLAs according to the system changes, in order to find a trade-off between the requests of both the layers.

As shown in Figure 1 the main PaaSSOA actors involved in the interactions between the PaaS and the IaaS layers are the Functions and the Resource Manager blocks. In fact, as depicted in Figure 2, the Function block is composed of a CDM block which has a Negotiator component which is in charge of communicating with the IaaS layer. As shown in Figure 4 instead, the Resource Manager is composed of a Resource Manager itself (RM) and the Monitor Agents (MAs). The RM is in charge of maintaining the status of the utilization of the physical resources and, according to the scheduler chosen, to allocate them to the virtual machines. The MA is in charge of monitoring a single physical resource and reports its status to the RM. Since the status of the IaaS infrastructure can be impacted by the performances of the physical resources as well as by the virtual ones, it is very important to monitor both using two different MAs: the Physical Monitor Agent (PMA) and the Virtual Monitor Agent (VMA).

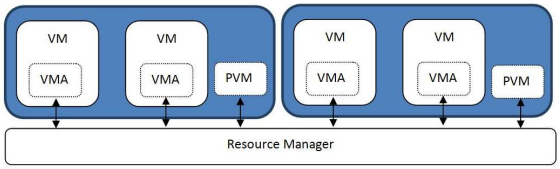

Figure 4: The IaaS Resource Manager Architecture.

The PMA leverages the resource functionalities in the intent of monitoring *Physical Metrics* such as Availability, Memory Available for VMs, Memory Used by VMs, CPU Used, CPU Idle, Number of CPUs.

The VMA leverages the hypervisor functionalities for the purpose of monitoring *virtualization metrics*, which include the following parameters: a) *VM Metrics* such as Availability, Process Virtual Memory Size, Process Resident Memory Size, Process Page Faults, Process CPU System Time, Process CPU User Time, Process Uptime, Process CPU Total Time, Process CPU Usage, VM CPU Wait, VM CPU Used, VM CPU Sys, VM Memory Shares, VM Memory Minimum, VM Memory Maximum, VM Memory Size, VM Memory Ctl, VM Memory Swapped, VM Memory Shared, VM Memory Active, VM Memory Overhead, VM Uptime; b) *VM Disk Metrics* such as Availability, Reads, Writes, Bytes Read, Bytes Written; c) *VM NIC Metrics* such as Availability, Packets Transmitted, Packets Received, Bytes Transmitted, Bytes Received.

All the information coming from the MAs will allow the RM to implement the best scheduling policy to achieve the optimal configuration of the system. For example, a proportional thresholding (H. Lim and Parekh, 2009) approach could be used to determine the target range of the variables according to which a reconfiguration event needs to be fired.

The prototype implementation of the system vision we presented in this paper, uses the VIDA toolkit (Anedda, 2011), a novel software architecture for the deployment of virtual clusters currently developed as a part of one of the research activities of one of the authors'. The VIDA architecture is designed to dynamically allocate resources to applications via a general control plane orchestrating the system's computational and network components. Hence, it meets our best expectation at the PaaS to IaaS interface.

4 CONCLUSIONS

In this paper we have illustrated the highlights of our vision for a new approach to the development of an open PaaS framework which captures the main functions which characterize the Cloud from the PaaS perspective. In our discussion we have highlighted the interactions that the PaaS layer ought to have with the two adjacent levels in the Cloud SPI stack in order to draw maximum benefit the mediator potential of its role.

By virtue of the four constituent blocks sketched in figure 1, we were able to clearly define and apportion the responsibilities of each component of PaaS architecture as well as to capture the interactions required among the different layers of the Cloud SPI stack. Moreover, for each block in the envisioned architecture we provided details on the inner working of the relevant components. We also sketched the communication interface to be put in place between the PaaS and the IaaS layers, outlining the main functionalities and parameters to flow across them.

We are currently developing a PaaSSOA prototype using the Jolie (Guidi and Montesi, 2011) technology: the demonstrator code may be downloaded at (Guidi, 2012).

ACKNOWLEDGEMENTS

We are grateful to Giovanni Giacobbi, member of our team at the University of Padova, for his suggestions and interesting discussions during the write-up of this paper.

REFERENCES

Anedda, P. (2011). *Virtual Infrastructures Deployment Architecture*. http://project-vida.sourceforge.net/.

EU (2011a). *Irmos - Interactive Realtime Multimedia Applications on Service Oriented Infrastructures*. http://www.irmosproject.eu/.

EU (2011b). *Reservoir - Resources and Services Virtualization without Barriers*. http://62.149.240.97/.

Google (2011). *Google App Engine*. http://appengine.google.com/.

Guidi, C. (2012). *PaaSSOA: Paas Service Oriented Architecture*. http://sourceforge.net/projects/paassoa/.

Guidi, C. and Montesi, F. (2011). *Jolie*. http://www.jolie-lang.org.

H. Lim, S. Babu, J. C. and Parekh, S. (2009). Automated control in cloud computing: Challenges and opportunities. In *First Workshop on Automated Control for Datacenters and Clouds*.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C. (2005). *Web Services Choreography Description Language Version 1.0*. W3C Candidate Recommendation 9 November 2005.

opencloudmanifesto.org (2009). *Open Cloud Manifesto*. http://www.opencloudmanifesto.org.